CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Ingeniería Informática

Universidad Politécnica de Madrid

Facultad de Informática

TRABAJO FIN DE GRADO

Nuevo algoritmo de relajación para motores de cuerpos sólidos.

Autor: Luis Enrique Muñoz Martín

Director: Alfonso Rodríguez-Patón Aradas

MADRID, JUNIO DE 2014

Índice de contenido

Índice de figuras

Resumen

Con el fin de conocer mejor a las bacterias, en la actualidad se han desarrollado aplicaciones que permite simular el comportamiento de las colonias formadas por este tipo de organismos. Una de las piezas más importantes que tienen estos simuladores es el motor de físicas. Éste es el encargado de resolver todas las fuerzas producidas entre las bacterias y conseguir que todas queden correctamente colocadas y distribuidas a lo largo de la colonia, tratando de asemejarse lo más posible a la realidad.

En una simulación de éstas características, todas las bacterias, además de estar en contacto entre sí, crecen en un pequeño porcentaje durante cada fotograma. Ello produce una gran cantidad de solapamiento a lo largo de toda la colonia que el motor de físicas tiene que resolver.

El trabajo que se describe en este documento surge de la ineficiencia del proceso actual para distribuir el solapamiento originado en el interior de la colonia, hasta su exterior. Es importante señalar que la física se lleva el 99% del tiempo de procesado de la simulación de una colonia, con lo que una mejora en el motor de físicas conseguiría incrementar en gran medida la capacidad de simulación. El objetivo no es otro que poder simular más cantidad de bacterias en menos tiempo, facilitando el estudio de esta área tan reciente como es la biología sintética.

Abstract

In order to better understand bacteria, new applications have been developed to simulate the behavior of colonies formed by these organisms. One of the most important parts of these simulators is the physics engine. This module is responsible for solving all the forces produced between bacteria and ensure that they are properly located and distributed throughout the colony, trying to be as close as possible to reality.

In a simulation with these features, all bacteria, besides being in contact with each other, grow in a small percentage at each frame. This produces a large amount of overlap along the entire colony that the physics engine must solve.

The work described in this document arises from the inefficiency of the current process to distribute the overlap originated at the core of the colony outwards. Importantly, physics takes up 99% of the processing time of the simulation of a colony. Therefore, improving the physics engine would translate in a drastic increase in the throughput of the simulation. The goal is simply to be able to simulate more bacteria in less time, making the study of the recent area, synthetic biology, much easier.

1. Introducción

Lo expuesto aquí no trata de transformar el funcionamiento de los motores físicos de cuerpos sólidos [1], sino dar una solución a un problema concreto que se origina en la actualidad en dichos motores. Este problema es de carácter computacional, es decir, para una determinada distribución espacial de los objetos, la cantidad de cómputo que requiere su resolución física es muy elevada. Concretamente origina cuando se quiere resolver las colisiones producidas entre una elevada cantidad de objetos, todos ellos situados en un espacio inferior al mínimo necesario para que permanezcan sin que se produzcan solapamientos.

Para explicar bien por qué es necesario tanto cómputo en esa situación, hay que entender el funcionamiento básico de un motor de física de cuerpos sólidos. Cuando dos objetos colisionan, se restauran sus posiciones mediante unos parámetros físicos asociados a estos. Este procedimiento funciona perfectamente para dos cuerpos aislados. Para un conjunto de cuerpos solapando entre sí, es muy probable que la resolución de una colisión individual, dé como resultado una posición no vacía para el objeto desplazado, es decir, la resolución de una colisión, puede generar otra colisión. La solución usada actualmente es aplicar más iteraciones de física por fotograma a ese conjunto de objetos, de forma que poco a poco los solapamientos originados desaparezcan al distribuirse dichos objetos hacia zonas más vacías.

En muchas de las aplicaciones que utilizan motores físicos de cuerpos sólidos, como simulación de coches, aeronaves o videojuegos, los casos antes mencionados se producen de manera pequeña y aislada, por lo que incrementando ligeramente el número de iteraciones de física realizadas por fotograma se soluciona el problema.

En cambio, en un simulador de colonias de bacterias [2] este problema se lleva al extremo: en cada fotograma las bacterias crecen una pequeña longitud, provocando que todas solapen con las de alrededor; además, las únicas fuerzas existentes provienen de las propias bacterias por lo que no hay nada que las haga desplazarse para ocupar las zonas vacías del exterior a no ser que sean empujadas por las vecinas. Esto provoca que la colonia nunca tenga más espacio del necesario para albergar a todas las bacterias de una forma relativamente compacta. En una situación así, cualquier resolución de un solapamiento que no pertenezca al contorno de la colonia provocará otro, ya que no hay espacio libre.

Ante esta peor situación descrita, lo que se realiza es aplicar muchas iteraciones de física, de manera que a “fuerza bruta” se consigue desplazar los solapamientos hacia el exterior de la colonia reduciendo todos los producidos en su interior. Como se explicará más detalladamente, el número de iteraciones necesarias se incrementa cuadráticamente respecto al radio de la colonia. Por si fuera poco, estas colonias de bacterias suelen crecer de manera exponencial, y por lo tanto el número de iteraciones de física necesaria para el desplazamiento de dichos solapamientos aumenta muy rápido, por lo que nuestra capacidad de simulación decae drásticamente.

El algoritmo que se explicará en este documento trata de paliar ese efecto y conseguir que los solapamientos de la colonia se resuelvan con un número de iteraciones constante e independiente del número de bacterias, es decir, disminuir el orden algorítmico actual de $O(n)$ a $O(n)$.

Éste ha sido desarrollado pensando en su funcionamiento dentro un motor de físicas de un simulador de colonias de bacterias orientado inicialmente a 2 dimensiones, por lo tanto a la hora de su descripción y validación nos referiremos a él en dicho contexto. Ello no descarta sus posibles usos en otros simuladores que no estén orientados a colonias de bacterias, ni su modificación a 3 dimensiones, ya que es independiente de la forma de los objetos y de la física que controla la interacción de estos.

También es relevante destacar que el algoritmo, por estructura, solo se ejecuta cuando sea necesaria una liberación de presión, y únicamente en la zona precisa. Por lo tanto, su implementación en un sistema ya desarrollado no alterará otras zonas que no requieran de su actuación.

Todo lo expuesto a en este documento no son simples indicaciones teóricas de lo que se podría hacer para mejorar el tiempo de cómputo. Todas las etapas de las que se compone el algoritmo de Expansión por anillos que se va a explicar, han sido implementadas y probadas en un motor de físicas, CellEngine, creado para la evaluación y la realización de las pruebas, además de como futuro motor de físicas orientado a simuladores de colonias de bacterias.

1.1. Estructura del documento

- Estado del arte. En él se explica la relevancia de los simuladores de colonias de bacterias, además de presentar varios simuladores que hay actualmente en uso, indicando las características más relevantes de éstos.

- Planteamiento del problema. Inicialmente se explica cómo funcionan a groso modo los motores de física de cuerpos sólidos, para luego poder entender más profundamente la solución aplicada actualmente. En este apartado también se desarrollan las matemáticas asociadas a la relajación de una colonia para posteriormente analizarlas y extraer las herramientas que se utilizarán en la solución planteada, ya que dicha solución se basa en la que hay actualmente.

- Solución planteada: Expansión por anillos. En primer lugar se explica un resumen a alto nivel de lo que consiste el algoritmo y de cómo obtenemos una reducción de n en el orden algorítmico. A continuación se pasa a describir más detalladamente las etapas de las que consta, para finalmente, comentar la relación calidad/velocidad que se puede obtener variando un determinado valor dependiente del algoritmo.

- Resultados y comparativas. En este apartado se muestran los resultados del nuevo algoritmo, sus ventajas y desventajas. Además, se utilizará el simulador de pruebas con el motor CellEngine para realizar comparaciones con varios

simuladores actuales. También se describen varias formas de validar los resultados.

- Conclusiones y futuros trabajos. Se realizará un repaso a lo investigado durante el documento, haciendo hincapié en los resultados conseguidos. Seguidamente se explicará de forma breve las futuras mejoras que se realizarán al algoritmo, y su incorporación a un simulador ya existente.

- Anexo A: Ejemplo de cálculo de un solapamiento. Este anexo es información complementaria al apartado de matemáticas dentro del planteamiento del problema. En él se realiza un ejemplo en el que se utilizan las relaciones halladas, con el fin de ayudar a su entendimiento.

- Anexo B: Algoritmos para la implementación de las etapas. En él se explican distintos procedimientos para la implementación de las etapas del algoritmo de Expansión por anillos, descritas en el apartado de etapas, de la solución planteada. También se comentan varias ideas sobre cómo paliar los posibles errores que pueden surgir a la hora de su implementación.

- Anexo C: Función troncal de CellEngine. En este anexo se analizará el código asociado a la función de más alto nivel correspondiente a la ejecución de un paso de física del motor.

2. Estado del arte

La biología sintética [3,4] es una ciencia relativamente moderna, ya que hasta hace unos pocos años no se conocían los "ladrillos" para el diseño de nuevos sistemas biológicos. Es por ello, que el uso de simuladores de colonias de bacterias no ha tenido una aplicación funcional hasta estos últimos años. Estos simuladores son los encargados de simular la interacción que se produce tanto entre las propias bacterias como entre las bacterias y las señales del medio. De esa manera, el biólogo puede realizar experimentos orientativos acerca de cómo afecta una determinada sustancia a un conjunto de bacterias o de cómo se propaga cierta señal entre ellas.

El tiempo necesario para permitir que una colonia crezca en una placa de Petri o *biofilm* [5] hasta el tamaño deseado para un experimento suele llevar entre 24 y 48 horas en un laboratorio. Muchas de las pruebas que se efectúan sobre una colonia la dejan inservible para otra posible prueba, y para realizar una investigación cualquiera son necesarios cientos de experimentos como estos. Por lo tanto una herramienta que simule dichos experimentos y que reduzca drásticamente ese tiempo es crucial para el desarrollo de cualquier investigación. Obviamente, los resultados de una colonia simulada no son tan reales como los realizados sobre una placa de Petri, pero sí que permite al biólogo descartar ciertos resultados claramente negativos en la búsqueda de su objetivo.

A estos simuladores también se los llama, modelos basados en individuos [6], ya que lo que se hace realmente es programar tanto la acción como la interacción a nivel de individuo. Esto entre la totalidad del conjunto de los individuos genera la colonia con sus características, características que a priori son desconocidas y no se han programado directamente.

Una de esas interacciones es la conjugación [7] cuya simulación tiene una gran importancia en los experimentos actuales. Consiste en la capacidad que tienen algunos tipos de bacterias en traspasar código genético a sus vecinas. Ese código genético que se traspasan modifica el comportamiento a nivel de las bacterias y por tanto, modifican los resultados generales del experimento.

Gran parte de lo que se considera un simulador de colonias de bacterias recae en el motor de físicas que este contenga. Un simulador de bacterias sin un motor de física no podría simular absolutamente nada, pero un motor de físicas individual ya podría darnos información relevante sobre ciertos fenómenos físicos ocurridos en una colonia bacteriana, de ahí su importancia.

Es además el encargado de resolver las colisiones a nivel de individuos y de la correcta colocación de estos en la colonia mediante un procedimiento de empuje. Este empuje consiste en desplazar el solapamiento que se va produciendo desde el interior hacia el exterior de la colonia, con el fin de relajar todas las fuerzas que están actuando en ella. Este procedimiento es clave en la simulación de una colonia y es el proceso que más cómputo se lleva de toda la simulación.

Para la simulación del cuerpo de una bacteria, se suele utilizar una forma capsular [8]. Cabe destacar que aunque las simulaciones con éste tipo de formas son muy fieles a la realidad, existen ligeras discrepancias debido a que las bacterias no son cuerpos sólidos como tales, y pueden deformarse y adaptarse ligeramente a las fuerzas que actúen sobre ellas. Sin embargo, son computacionalmente rápidas comparándolo con otro tipo de aproximaciones al tomarse éstas como el lugar geométrico a una determinada distancia de un segmento. La rapidez en la colisión es vital ya que se ejecutará cientos de miles de veces por fotograma.

A continuación se detallan algunos simuladores de colonias de bacterias que tienen cierta relevancia en la actualidad.

2.1. Gro

Gro [9] es un simulador 2D de micro-colonias, desarrollado por el laboratorio de Klavins de la Universidad de Washington y de libre distribución. Lo característico de este simulador es que permite el uso de un lenguaje de muy alto nivel, desarrollado por ellos mismos, para describir y modelar comportamientos bacterianos, aislando al biólogo de la complejidad de la programación de menor nivel, además de permitir la traslación del experimento real al simulado con unas pocas líneas de código. El simulador es considerado de *prototipado*, aunque sus resultados son bastante fieles a los sucesos de la realidad.

Para la realización de la física, gro utiliza un motor físico de cuerpos sólidos y de libre distribución, llamado Chipmunk [10], desarrollado por la compañía Howling Moon Software. Este motor está orientado a juegos, por lo que no está especialmente enfocado a simular la física entre bacterias. Por un lado, no existe la física entre formas capsulares, que es lo más fiel a la realidad y más rápido de computar para estos casos. Lo que se utiliza en su lugar son polígonos ortogonales tratando de imitar la forma capsular que éstas tienen. Por otro lado, no utiliza hardware adicional para el cómputo de la física (como GPUs), que es lo que se lleva casi toda la carga de trabajo en estas aplicaciones. Estos motivos Gro especifica que su simulador está orientado a colonias de un tamaño reducido. Una imagen de éste simulador la podemos ver en la figura 1.

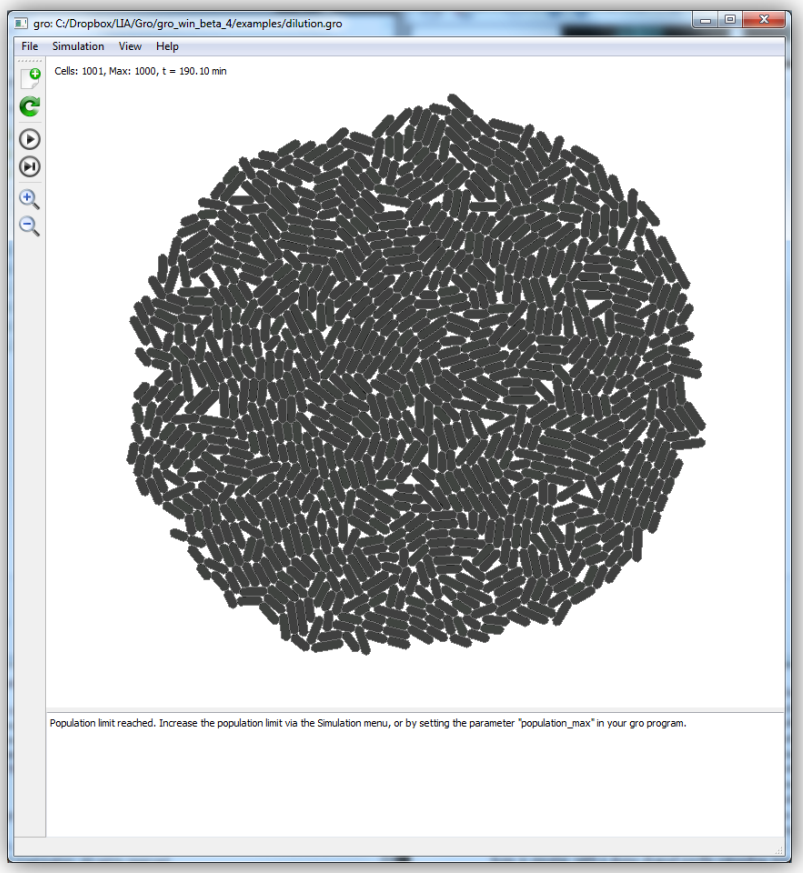

Figura 1. Simulador Gro

2.2. CellModeller

CellModeller, mostrado en la figura 2, es el pionero en su sector [11]. Ha sido desarrollado por el laboratorio de Haseloff de la Universidad de Cambridge y forma parte de Microsoft Research, equipo de investigación financiado por Microsoft. Permiten realizar simulaciones de colonias tanto en 2D como en 3D con una gran similitud con la realidad.

El motor físico lo han programado exclusivamente para su simulador. Está enfocado a la simulación de formas capsulares para su funcionamiento en colonias de bacterias. Además utilizan GPUs para acelerar el rendimiento, asegurando conseguir 32000 bacterias en 30 minutos.

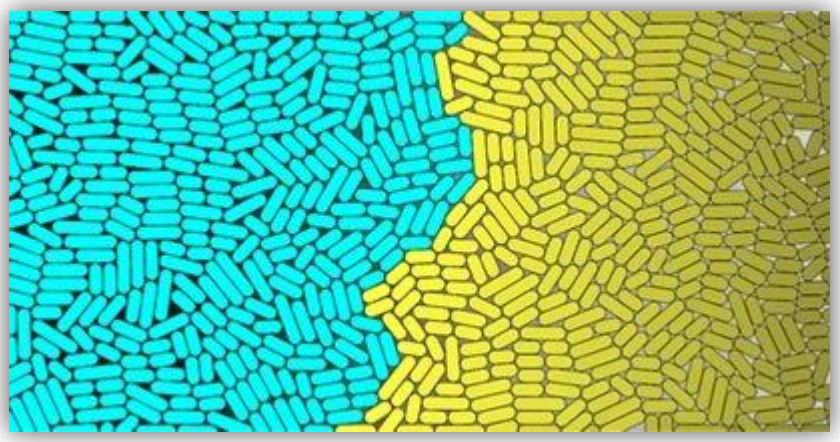

Figura 2. Simulador CellModeller

2.3. BSim

BSim [12] es un simulador de desarrollado por La Universidad de Bristol. Es de carácter general, es decir, está destinado a distintos tipos de simulaciones, puede simular desde partículas en un fluido 3D a bacterias sobre una placa de Petri, como se puede apreciar en la figura 3. Permite cargar externamente la forma del objeto con la que se quiere realizar la simulación, permitiendo una gran escalabilidad.

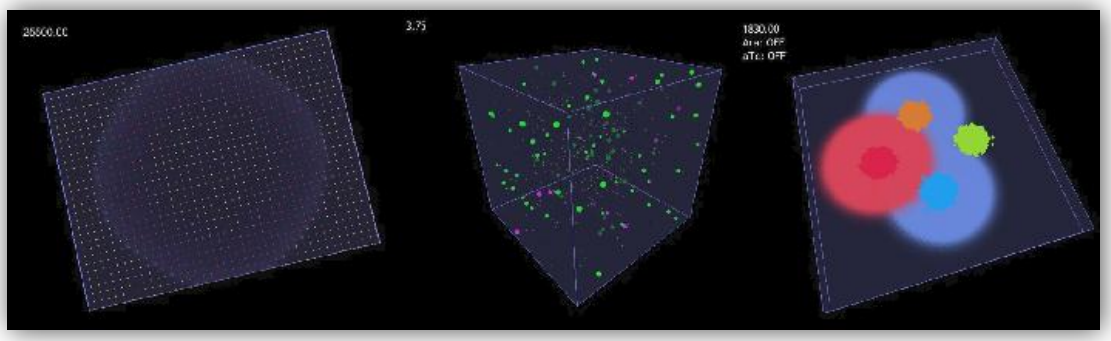

Figura 3. Simulador BSim

2.4. iDyNoMICS

iDyNoMICS [13] es un simulador desarrollado por la universidad de Birmingham, orientado simular una gran cantidad de bacterias, tanto en 2D como en 3D. Para ello trata a las bacterias como si fueran esferas, que son formas geométricas muy rápidas de computar. Es considerado como una herramienta de prototipado, con la que realizar las simulaciones en busca de unos primeros resultados. Una característica relevante de este simulador es que está desarrollado como *framework* para la simulación de bacterias, permitiendo extensiones como la forma capsular, conjugación celular, etc. En la figura 4 se puede apreciar una imagen de este simulador, el cual muestra una colonia tomando como eje x el *biofilm.*

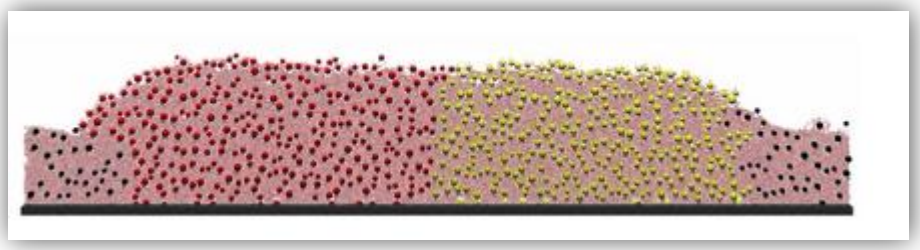

Figura 4. Simulador iDyNoMICS

2.5. BactoSim

BactoSim [14] es un simulador desarrollado por el departamento LIA, de la UPM como parte del proyecto europeo PLASWIRES [15]. Está, sobretodo enfocado a la conjugación y a la propagación del código genético a través de la colonia. Las células se modelan como círculos sobre una malla discreta con el consiguiente ahorro de cálculo, permitiendo observar la propagación del código genético a través de la colonia. Se considera una herramienta de prototipado.

3. Planteamiento del problema

En los experimentos reales, las pruebas se hacen sobre colonias del orden de 10^9 - 10^{10} individuos. Estos experimentos se realizan con distintos tipos de bacterias, aunque por lo general, se suele utilizar la E. Coli, una de las más estudiadas. Esta bacteria tiene un tiempo medio de división de 20 minutos [16]. Este valor temporal real, pone un límite a los simuladores de bacterias. Podremos simular un número de bacterias tal que el tiempo que tarda en simularse la siguiente división sea menor que 20 minutos. Dicho de otra manera, una vez que tardemos el mismo tiempo en realizar un experimento en la realidad que en una simulación, esta última pierde gran parte de su sentido. Recordemos que el objetivo principal de simular un experimento es el de reducir su tiempo, y de esa manera poder estudiar más rápidamente el objetivo en cuestión.

En el simulador Gro, dicho límite, en la que la simulación va más lenta que en la propia realidad, lo alcanzamos con una población de 7000 individuos, y en el caso de CellModeller, lo alcanzamos con 32000 individuos. Estos límites se alcanzan con un ordenador estándar de sobremesa, que pueda tener el biólogo fácilmente accesible para realizar sus experimentos.

Si observamos las cifras, podemos empezar a suponer donde está el problema. En Gro, con un solo hilo de ejecución y sin un motor orientado a la física entre bacterias, el límite es tan solo 4.5 veces inferior que el de CellModeller, que sí utiliza un motor con formas capsulares además de cientos de hilos para la física. Está claro que hay un cálculo que se agrava drásticamente conforme aumenta el número de individuos.

Para entender qué es lo que pasa exactamente, hay que indagar en el funcionamiento básico de un simulador de físicas de cuerpos sólidos. Cuando dos objetos colisionan, la colisión se resuelve mediante los parámetros físicos asociados a ellos (masa, velocidad, etc.) y por la cantidad de solapamiento producido en la colisión. El resultado, tras su resolución, son dos objetos libres de dicho solapamiento, ya que los cuerpos se han desplazado hacia distintos lugares. Esto ocurre para dos únicos objetos en el espacio. En cambio, cuando hay más de dos, es posible que la resolución de una colisión determine como destino de un objeto un lugar ya contenido por otro. Ello provocaría otro solapamiento o colisión que hay que resolver.

En una colonia de bacterias, esto pasa prácticamente siempre al estar todas en contacto. Cualquier resolución de un solapamiento producido entre dos bacterias genera otra. El único caso que se escapa de esta situación es la resolución de una bacteria al borde de la colonia, la cual es empujada hacia el exterior, donde no hay nadie. ¿Cómo se resuelve esta situación?

3.1. Solución actual

La forma de resolver este problema, utilizada tanto por Gro como por CellModeller, es aplicar un algoritmo de relajación que consiste simplemente en ejecutar iteraciones de física, hasta que poco a poco el solapamiento se va distribuyendo hacia el exterior.

Cuando hablamos de una iteración de física a la colonia nos referimos a:

- Una primera pasada a la población en la que se hallan las resultantes de las fuerzas que intervienen en cada una de las bacterias con las de su vecindad inmediata.

- Un segundo recorrido en el que se posiciona y rota a todas las bacterias según su fuerza resultante.

Por tanto, cabe señalar, que una iteración de física es de $O(n)$ respecto al número de bacterias o individuos de la colonia.

En la siguiente sección, se estudiará cómo incrementa el cómputo del algoritmo de relajación según el número de bacterias, y cuantas iteraciones de física son necesarias respecto a éstas.

3.1.1. Funcionamiento

Inicialmente en cada fotograma, todas las bacterias incrementan en un determinado tamaño su longitud, por lo que generan un solapamiento con sus vecinas, provocando que la colonia entera acabe con un porcentaje de solapamiento homogéneo. Cabe destacar que en el crecimiento de una colonia, el área de ocupación se propaga hacia todos los sentidos de manera equidistante generando una forma circular, cada vez más perfecta según incrementa el número de individuos. Por ello, consideraremos a la colonia como si de un círculo se tratase en cuanto a su morfología.

En la resolución de una bacteria, ésta se desplaza según su fuerza resultante, que vendrá dada por los solapamientos con sus vecinas. En el interior de la colonia, una solución de este estilo generará otro solapamiento, ya que no hay espacio para alojar a la bacteria en la nueva posición. Aparentemente parece que no se soluciona nada, simplemente desplazamos (sin saber a dónde) el solapamiento de un sitio a otro de la colonia. El único caso en que sí se solucionaría algo es cuando una de las dos bacterias pertenece al borde, ya que su resolución ensancharía los límites de la colonia y crearía espacio para los demás solapamientos originados.

Para estudiar lo que pasa realmente supondremos que todos los individuos son estadísticamente iguales y que todos están solapando en un primer instante. Veamos que ocurre al ir aplicando iteraciones. En la figura 5 se muestra una colonia trasladada a una dimensión y mostrada sólo desde su centro al exterior. Nos referiremos a los diferentes individuos dependiendo del nivel (L) en el que estén, de menor a mayor contando desde el exterior.

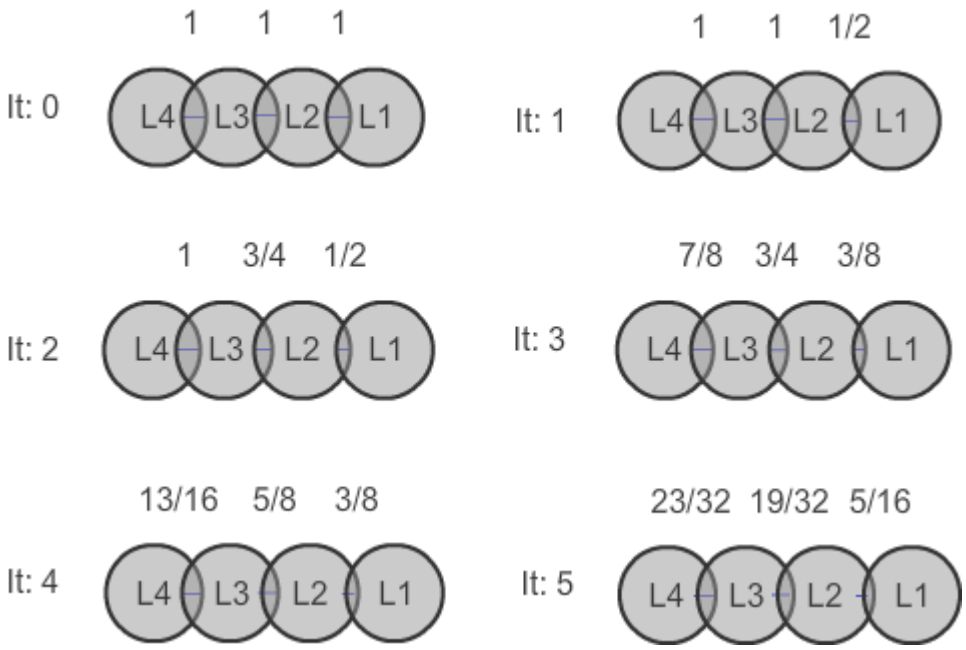

Figura 5. Colonia 1D. Funcionamiento actual.

Los solapamientos originados únicamente se puede resolver en el borde de la colonia (L1), y por lo tanto el desplazamiento de L1 es el único que genera espacio en la colonia. Dicho espacio generado, permite el desplazamiento del solapamiento más interno. Observamos que el solapamiento en el borde de la colonia (L1), tiende a solucionarse más o menos rápido en unas pocas iteraciones. Mientras que el solapamiento en el interior (L4) tiene que ir desplazándose poco a poco hasta el borde para ser solucionado. Cabe destacar que L4 se considera "anclada" ya que sus resultantes durante toda la simulación serán 0 al estar en el centro de la colonia.

La lentitud de la resolución de los solapamientos, proviene del poco valor que tiene las fuerzas resultantes generadas entre los distintos individuos. Las fuerzas de oposición originadas son casi idénticas a las de empuje, agravándose aún más en los solapamientos cercanos al centro de la colonia. Analicemos las matemáticas que hay detrás de esto.

3.1.2. Matemática del empuje

Un solapamiento se resuelve como fuerzas hacia ambos lados. La magnitud de cada una de las fuerzas es la mitad de ese solapamiento. Por lo tanto, para determinar cómo varía el valor de uno en concreto $s_{i,d}$ (para una determinada iteración i y a una determinada distancia del borde d), tenemos que ver los desplazamientos o fuerzas que los vecinos inmediatos hacen sobre él, el cual reducido a una dimensión, seguiría el siguiente esquema:

- El individuo situado en la posición $d + 1$ (más cercano al centro) añadirá el siguiente solapamiento al ya existente:

$$s_{i,d} = s_{i,d} + \frac{s_{i-1,d+1} - s_{i-1,d}}{2}$$

- El individuo situado en la posición $d - 1$ (más cercano al borde de la colonia) reducirá solapamiento al ya existente:

$$s_{i,d} = s_{i,d} + \frac{s_{i-1,d1} - s_{i-1,d-1}}{2}$$

- Los solapamientos $s_{i,d}$ para valores de d fuera del intervalo $[1, r - 1]$, se consideran nulos. Los que valores superiores a la longitud del radio no ejercen fuerza de oposición y valores inferiores no ejercen fuera de empuje al no existir como tales.

Eso significa que el solapamiento varía según:

$$s_{i,d} = s_{i,d} + \frac{s_{i-1,d+1} - s_{i-1,d}}{2} - \frac{s_{i-1,d1} - s_{i-1,d-1}}{2}$$

Que simplificando obtenemos:

$$s_{i,d} = \frac{s_{i-1,d+1} + s_{i-1,d-1}}{2}$$

Veamos cómo evoluciona el desarrollo de $s_{i,d}$ para valores dependientes de iteraciones anteriores:

- Para $i - 2$:

$$s_{i,d} = \frac{s_{i-2,d+2} + s_{i-2,d-2}}{4} + 2\frac{s_{i-2,d}}{4}$$

- Para $i - 3$:

$$s_{i,d} = \frac{s_{i-3,d+3} + s_{i-3,d-3}}{8} + 3\frac{s_{i-3,d+1} + s_{i-3,d-1}}{8}$$

- Para $i - 4$:

$$s_{i,d} = \frac{s_{i-4,d+4} + s_{i-4,d-4}}{16} + 4\frac{s_{i-4,d+2} + s_{i-4,d-2}}{16} + 6\frac{s_{i-4,d}}{16}$$

De esto podemos observar varias cosas. Llamando k al número de iteraciones anteriores de las que depende, vemos que:

- Un solapamiento depende de los solapamientos producidos a k distancia de él al cabo de k iteraciones.

- En iteraciones impares, un solapamiento depende de todos los solapamientos vecinos a distancias impares y en distancias pares, depende de todos los solapamientos vecinos a distancias pares, incluyendo el mismo.

- Tenemos un factor común de la forma:

$$\frac{1}{2^k}$$

- Multiplicando la secuencia por 2^k, los valores de los coeficientes son los valores encontrados en la fila k de un Triángulo de Pascal (empezando a contar desde 0).

A partir de estas observaciones podemos sacar una versión generalizada para la dependencia de las k iteraciones anteriores:

$$s_{i,k,d} = \frac{t_{k,k} \cdot s_{i-k,d+k} + t_{k-2,k} \cdot s_{i-k,d+k-2} + \cdots + t_{k-2,k} \cdot s_{i-k,d-k+2} + t_{k,k} \cdot s_{i-k,d-k}}{2^k}$$

Siendo $t_{x,y}$ el valor a distancia x de la columna central, e y la fila de un Triángulo de Pascal con la representación mostrada en la figura 6.

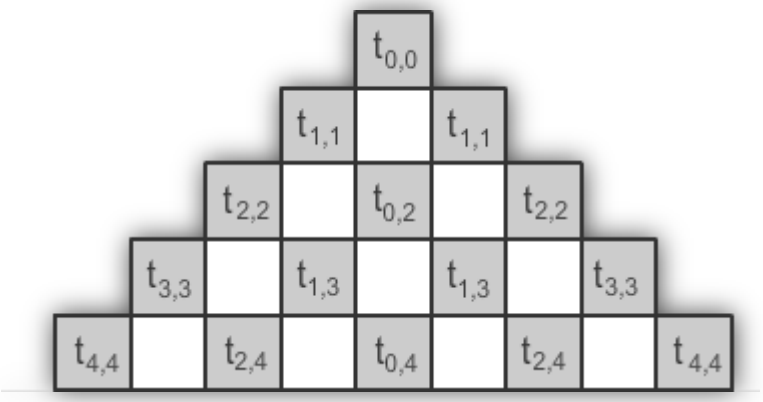

Figura 6. Triángulo de Pascal

A partir de esta representación podemos darnos cuenta de que la resolución de un solapamiento depende de todos los demás individuos de la colonia conforme se incrementa el número de iteraciones. Para una iteración k, el radio de efecto es k aunque no todos los individuos ejercen fuerza en una misma iteración.

Gracias a éste desarrollo genérico también podemos observar que individuo a distancia x ejerce fuerza sobre otro individuo al cabo de x iteraciones. Además, gracias a la distribución de coeficientes del Triángulo de Pascal podemos observar con qué porcentaje de fuerza actúa un individuo a x distancia. Llamaremos F a la función que determina la fuerza que un individuo realiza sobre otro a una distancia d y tras k iteraciones (teniendo los valores d y k, un valor par de separación entre ellos):

$$\mathrm{F(d,k)} = \frac{t_{d,k}}{2^k}$$

Para obtener el valor de $t_{d,k}$ se ha de construir el piso del Triángulo de Pascal hasta el nivel $k - 1$, y a partir de ahí, realizar la suma de $t_{d+1,k-1}$ y $t_{d-1,k-1}$. Es importante destacar que en el proceso se pueden encontrar valores de $s_{i,k,d}$ cuya d esté fuera del intervalo $I = [-r + 1, r - 1]$, es decir, fuera del radio de la colonia, y por lo tanto sus solapamientos a partir de esa distancia serán nulos. El procedimiento para obtener una $t_{x,y}$ en un Triángulo de Pascal acotado se puede observar en el Anexo A: Ejemplo de cálculo de un solapamiento.

Para solapamientos que estén a rodeados de otros del mismo valor a una vecindad de radio k individuos, al cabo de k iteraciones las relaciones de fuerzas que cada uno de ellos realizará, vendrán dadas por los coeficientes de la línea k de un Triángulo de Pascal normal, y por lo tanto seguirán una distribución normal. La suma de todos ellos dará 1, es decir que la resultante de todas esas fuerzas es 0, y el solapamiento se conservará intacto. Por lo tanto, para que se resuelva un solapamiento en la iteración k, es necesario que haya alguno resuelto a una distancia de k individuos.

Suponiendo que el valor de solapamiento inicial s_0 está homogéneamente distribuido a lo largo del radio de la colonia podemos calcular fácilmente el que tendremos al cabo de k iteraciones desde el inicio del fotograma y a una distancia d del centro:

$$s_{k,d,r} = s_0 \frac{t_{k,k} + t_{k-2,k} + \cdots + t_{k-2,k} + t_{k,k}}{2^k} \quad \text{para } [-\mathrm{r} + 1, \mathrm{r} - 1].$$

Por un lado, vemos que hemos perdido la dependencia de la iteración i, esto es así gracias a que ahora tomamos como referencia el estado inicial s_0. Por otro lado, parece que también hemos perdido la dependencia respecto a la distancia. Esto sin embargo es una falsa apariencia, ya que el solapamiento homogéneo lo es únicamente dentro de los límites de la colonia, y por lo tanto, los valores de $t_{x,y}\ \forall x < k, \forall y < k$ fuera de dicha colonia han de tener el valor 0. Ello lleva a establecer una dependencia entre la x de $t_{x,k}$ y el radio de la colonia, para valores de $t_{x,k}$ que cumplan $d - x > 0\ y\ x - r < r$.

Suponiendo nuevamente que el solapamiento inicial es homogéneo, podemos hallar el porcentaje de solapamiento respecto al inicial que hay en un determinado lugar. Esta métrica nos es de gran utilidad ya que sirve para determinar cómo ha progresado la resolución del solapamiento a cierta distancia del exterior y al cabo de k iteraciones:

$$S(k,d,r) = \frac{s_{k,d,r}}{s_0} = \frac{t_{k,k} + t_{k-2,k} + \cdots + t_{k-2,k} + t_{k,k}}{2^k}$$

Un valor de $S(k,d,r) = 1$ significará que no se ha resuelto nada de solapamiento, y un valor de $S(k,d,r) = 0$ significará que se ha resuelto todo. Cabe señalar que la relación de S se puede ver como la última fila de un Triángulo de Pascal acotado, entre la última

fila de un Triángulo de Pascal normal. Ambos con k números de filas (empezando a contar desde 0).

Para saber cuál es el número de iteraciones que necesitamos para resolver un determinado porcentaje de solapamiento S, tenemos que despejar k, de la función $S(k,d,r)$. Llamaremos $K(S,d,r)$ a dicha función, la cual puede ser implementada algorítmicamente de manera sencilla, calculando filas del Triángulo de Pascal acotado hasta dar con una cuyo resultado dividido entre 2^i sea igual o inmediatamente superior al S dado.

Otra métrica interesante es el desplazamiento producido en la colonia al cabo de k iteraciones. Ésta vendrá dada únicamente por la resolución del solapamiento a distancia 1 del borde ya que es el único solapamiento que puede crear espacio para la colonia:

$$D(k) = \sum_{i=1}^{k} \frac{s_{i-1,1,r}}{2}$$

Sabiendo cuánto ha variado el radio de la colonia podemos determinar el área de solapamiento resuelta al cabo de k iteraciones que denominaremos como:

$$Ar(k,r) = \pi r^2 - \pi(r - D(k))^2$$

Suponiendo que el solapamiento al inicio de un fotograma está homogéneamente distribuido, podemos hallar el área inicial solapada fácilmente mediante:

$$As(s0,r) = \pi r^2 \cdot s_0$$

Esto nos permite calcular una bonita relación dada por $AS(s0,r,k)$ y que nos indica el porcentaje de solapamiento que queda por solucionarse en toda la colonia:

$$AS(s0,r,k) = \frac{Ar(k,r)}{As(s0,r)}$$

3.1.3. Análisis

En primer lugar, vamos a analizar como varía el porcentaje de solapamiento $S(k,d,r)$ según se incrementa el número de iteraciones k, en las figuras 7 y 8. Rápidamente podemos observar que sólo se obtiene una completa resolución con infinitas iteraciones.

Si nos fijamos más detenidamente, vemos que en un principio el solapamiento se resuelve relativamente rápido, siempre y cuando esté más o menos cerca del borde y en radios pequeños. Hay un valor, en la derivada de $S(k,d,r)$ cuando es mayor a -1, en la que la velocidad de resolución cambia de una manera bastante brusca, complicándose bastante dicha resolución. Este valor podría servir como indicador para saber cuándo es un buen momento de detener el número de iteraciones, ya que a partir de dicho valor el solapamiento resuelto en cada iteración es considerablemente pequeño.

Lamentablemente, ese umbral da distintos porcentajes de resolución dependiendo de la distancia al radio respecto a la que se mida, y en colonias relativamente grandes,

inevitablemente tendremos que realizar iteraciones por encima de ese límite, si queremos desplazar el solapamiento del interior. Además, se puede comparar entre las dos gráficas que al incrementar el radio al cuadrado, el número de iteraciones necesarias para resolver el mismo solapamiento aumenta drásticamente.

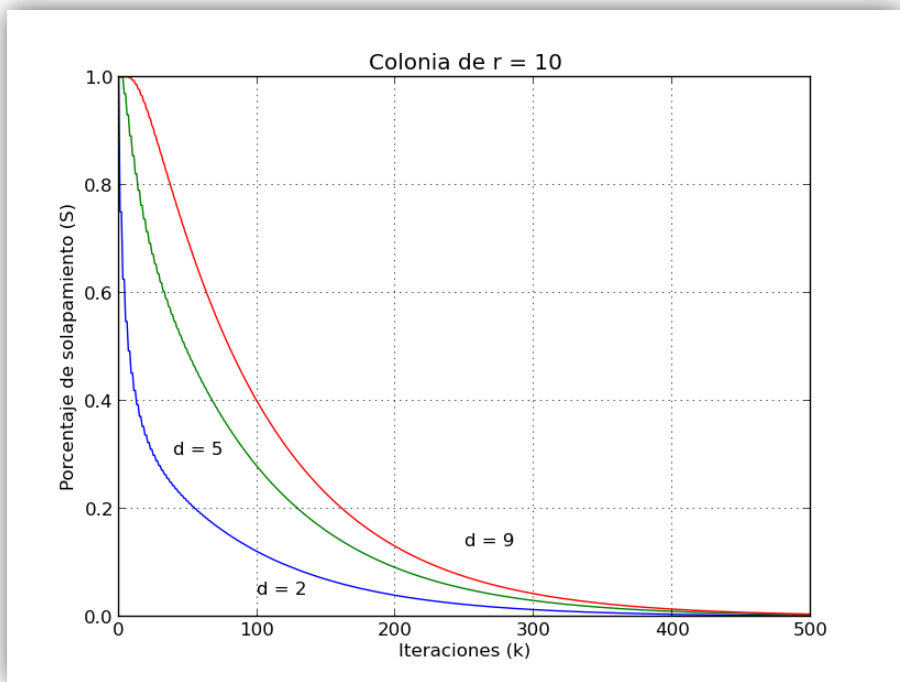

Figura 7. Porcentaje de solapamiento en relación a las iteraciones. r = 10

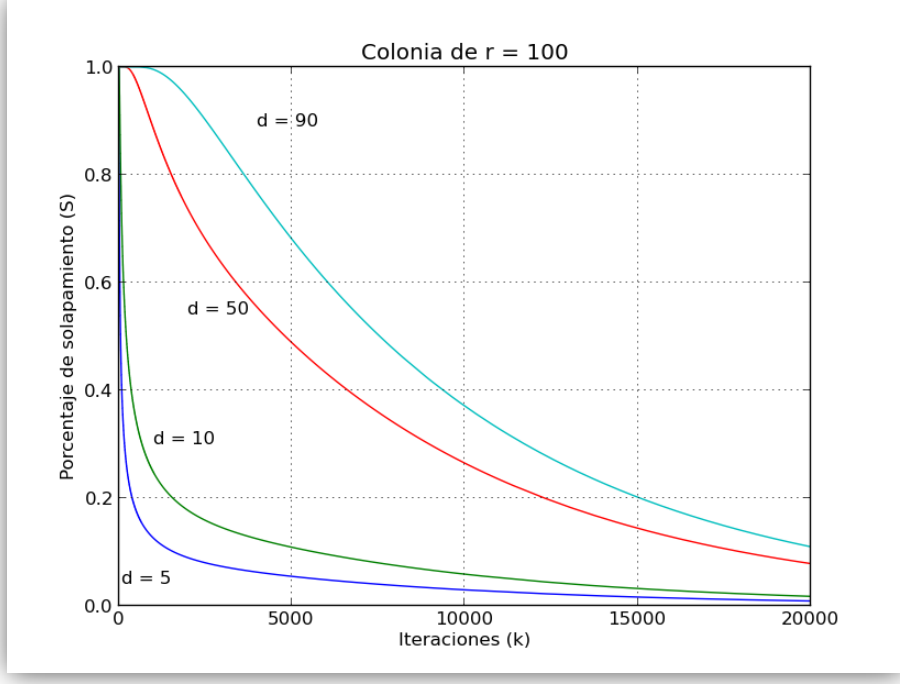

Figura 8. Porcentaje de solapamiento en relación a las iteraciones. r = 100

En la figura 9, se muestra la que probablemente sea la relación de más importancia, y que nos dice cómo aumenta el número de iteraciones necesarias conforme se incrementa la distancia al borde de la colonia $K(S, d, r)$. Para ello, suponemos tres valores fijos de porcentaje de solapamiento S para los cuales el solapamiento se considera resuelto a una distancia d del radio.

Primeramente se puede observar que tenemos unos valores discretos esto es porque la distancia al borde está contada en individuos discretos. Vemos que el valor de porcentaje resuelto incrementa el costo computacional de una manera bastante considerable. Respecto al número de iteraciones se puede apreciar que conforme nos alejamos del borde de la colonia esta aumenta cuadráticamente, ya que como hemos analizado, todas las bacterias dependen de la fuerza de sus vecinas a distancia k.

Esto nos lleva a pensar que el cómputo a partir de iteraciones suficientes como para que un solapamiento dependa de todos los demás de la colonia, debería empezar a reducir, ya que a partir de un determinado momento, el número de dependencias se mantendría constante, es decir, hemos llegado al punto en el que todas las bacterias ejercen fuerza sobre todas. Este efecto se puede apreciar en la función de color rojo cuyo porcentaje de solapamiento es de $S = 0.9$, al tener una resolución relativamente rápida.

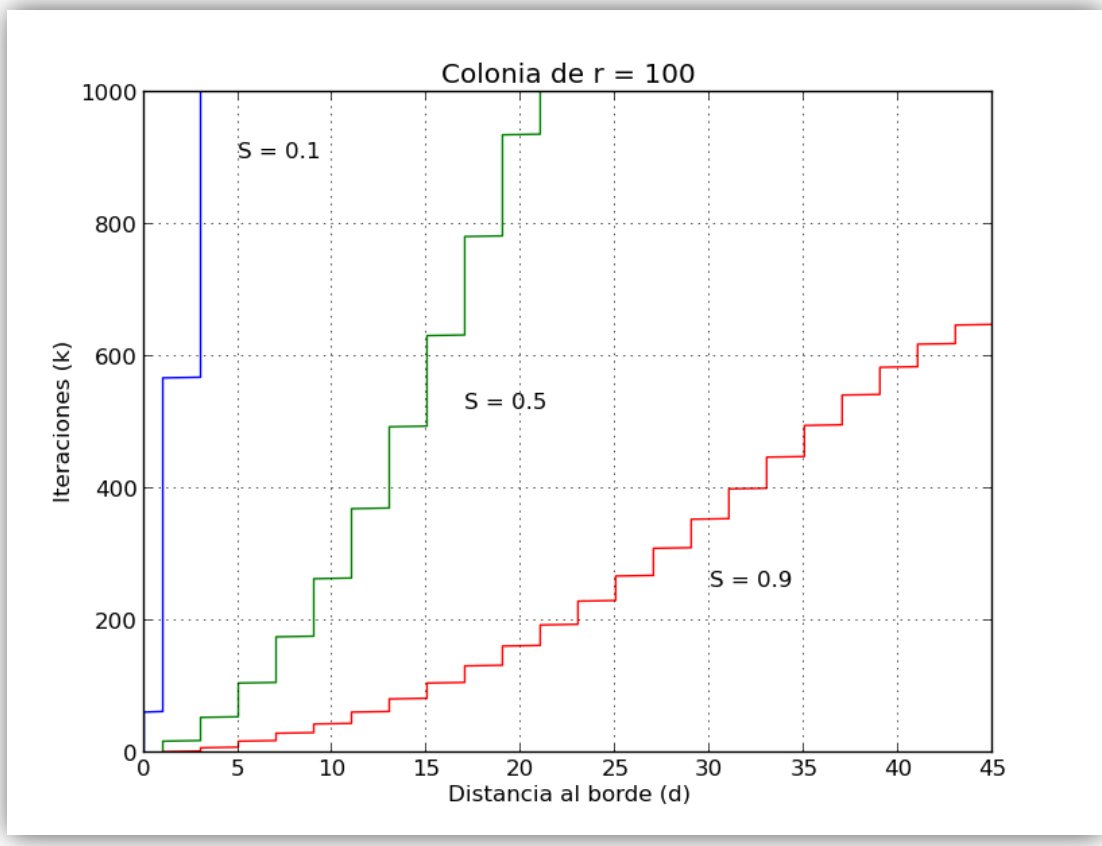

Figura 9. Iteraciones necesarias relativas a la posición del individuo

Veamos por tanto, en la figura 10, como afecta el radio para $S = 0.9$:

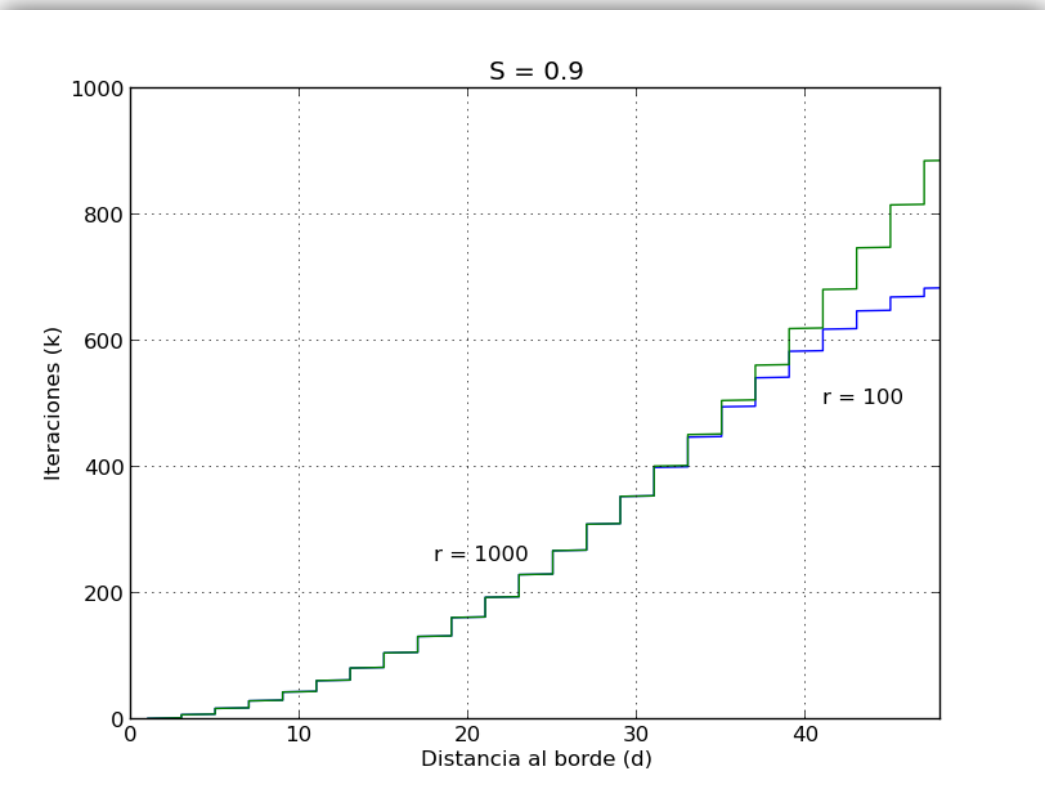

Figura 10. Comparativa de iteraciones necesarias según el radio de la colonia.

Esto nos viene a decir que no es lo mismo solucionar un solapamiento a x distancia del borde de una colonia grande que de una pequeña. Es por ello, que el radio de la colonia afecta también de una manera relativamente considerable a la rapidez de cómo se soluciona el solapamiento.

3.2. ¿Cómo afecta el funcionamiento descrito al cómputo de las simulaciones?

Lamentablemente, todo lo descrito solo funciona si consideramos que al inicio del fotograma todo el solapamiento es homogéneo. En un caso real, aunque todas las bacterias crecen la misma longitud y generan un solapamiento constante a lo largo del radio de la colonia, la resolución de solapamiento no lo es, por lo que generará cierta variación de área solapada a lo largo del radio. Esto incrementará aún más la dependencia cuadrática de las iteraciones respecto a la distancia al borde.

Sin embargo no es tan grave. Como en cada fotograma el solapamiento aumenta homogéneamente gracias el crecimiento estadísticamente homogéneo de las bacterias, aunque la distribución de la resolución no lo sea, el solapamiento al cabo de varios fotogramas se mantiene más o menos constante a lo largo del radio de la colonia.

Esto ocurre siempre y cuando seamos capaces de resolver medianamente bien el solapamiento, es decir, de aplicar suficientes iteraciones como para que el solapamiento del interior pueda llegar a "escapar" de la colonia. Mínimo necesitamos que $k = r$.

Dependiendo de cuantas iteraciones se realicen a partir de ese valor, el solapamiento podrá "escapar" más o menos rápido, y menos cantidad de solapamiento aproximadamente constante habrá a lo largo del radio de la colonia.

Normalmente en las simulaciones el crecimiento de las bacterias por fotograma es considerablemente reducido, por lo que aunque acabemos con un solapamiento del 500% (más o menos constante), apenas se notaría visualmente, ni provocaría discrepancias físicas.

Por lo tanto tenemos dos valores con los que jugar para controlar la densidad de solapamiento y la variación de éste a lo largo de la colonia: el número de iteraciones k y el crecimiento estadístico de cada individuo tras en cada fotograma. Un alto nivel de k, reducirá más rápidamente el solapamiento, y un alto nivel de crecimiento por fotograma lo mantendrá constante a lo largo de radio.

Una forma de determinar el solapamiento final que podemos obtener, es comprobando que:

$$S(k, r-1, r) + \frac{e}{s_{0,j-1}} \leq 1$$

Donde e, es el crecimiento homogéneo de los individuos en un fotograma y j el fotograma en ese instante. Esta relación viene a decirnos que: cuando la cantidad resuelta en el solapamiento del interior de la colonia (donde estará el mayor valor) más el crecimiento de los individuos, sea menor que el valor inicial de solapamiento que había antes de haber aplicado el último fotograma, habremos llegado al límite en el que el solapamiento se mantendrá constante.

Además como el solapamiento al inicio de la simulación, es producido por el crecimiento de la población, tenemos que $e = s_{0,j-1}$. Esto nos sirve para poder hallar ese límite desde un inicio, si es que existe.

Por lo tanto, podemos seguir considerando las gráficas expuestas como válidas en cuanto a proporciones y linealidades.

Como el radio es directamente proporcional al tamaño de la colonia $r = \sqrt{(n/\pi)}$, las iteraciones necesarias para relajar una colonia de n individuos son linealmente proporcionales dicho n. Esta nos permite saber, para una colonia dada, cuantas iteraciones tenemos que aplicar para conseguir un porcentaje de solapamiento que consideremos medianamente distribuido y con un valor relativamente. Notemos que el ideal sería obtener una $k = 1$, valor que nunca obtenemos, y que viene a decirnos que al computar únicamente una vez un individuo obtenemos su posición y orientación correcta para que sólo quede solapado un porcentaje de solapamiento que tomamos como aceptable y se mantiene constante durante toda la ejecución.

Ya que sabemos que una iteración de física es de $O(n)$, es decir se recorre una vez cada bacteria, y que k depende de una relación de lineal respecto al número de bacterias al depender cuadráticamente del radio, el algoritmo de relajación actual es de $O(n^2)$.

Es preciso recordar que el número de individuos aumenta exponencialmente, ello junto a un algoritmo de $O(n^2)$ provoca que para un tiempo de simulación relativamente pequeño, se estarán ya realizando grandes cantidades de cómputo, llegando muy rápidamente al tiempo de división real.

3.3. ¿Por qué se utiliza?

En primer lugar, es un algoritmo muy fácil de implementar, como hemos visto consiste simplemente en aplicar más iteraciones de física por fotograma.

Parece extremadamente costoso, ya que aproximadamente solo $1/r$ de las resoluciones de solapamiento (y cada vez en menor cantidad), están generando espacio en la colonia. Sin embargo, es bastante fidedigno a lo que ocurre en la realidad. Recordemos que la posición y rotación de una bacteria, depende de la colocación de todas las demás bacterias de la colonia, por lo que el algoritmo perfecto, tendría la complejidad de $O(n^2)$. Esto es así porque en base a aplicar iteraciones estamos encontrando el valor medio que debería tener cada fuerza.

A priori se podría pensar que la resolución de los solapamientos internos no aporta prácticamente nada, sin embargo esto no es totalmente cierto. Al solucionar dichos solapamientos se están creando fuerzas de oposición al desplazamiento, impidiendo que la colonia se expanda, fuerzas que provocan ciertos giros en las bacterias tratando de que se adhieran a cualquier pequeño espacio que pudiese haber. Estas fuerzas son además, las que crean el efecto neumático [17, 18] para formas capsulares como lo son las bacterias. Este efecto tiende a alinear a grupos de bacterias en la misma dirección, y es un signo de que los resultados se acercan a la realidad física. Lo podemos ver en la figura 11.

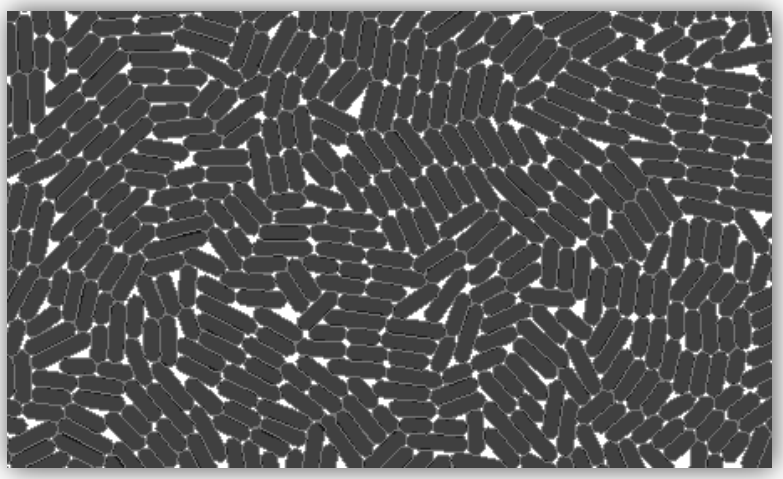

Figura 11. Efecto neumático. Imagen realizada a partir del simulador Gro

4. Solución propuesta: Algoritmo de Expansión por anillos

Como se ha visto, el algoritmo de relajación utilizado por los simuladores actuales, tiene una gran dependencia del número de individuos, los cuales crecen de manera exponencial. A continuación, se propone una forma para reducir dicha dependencia perdiendo el menor paralelismo con el suceso físico real.

4.1. Estudio de la solución

Mínimo, el algoritmo buscado ha de tener $\Omega(n)$ ya que necesitamos recorrer por lo menos una vez a cada bacteria de la colonia. Matemáticamente, dicho algoritmo constaría de una aplicación f a cada individuo de la colonia del siguiente estilo $b'_i = f(b_i)$, la cual modificaría la posición y orientación b_i de un determinado instante, en b'_i. Aplicando este procedimiento a toda la colonia conseguiríamos relajarla en $O(n)$, por lo que en un principio parece viable. El problema reside en si f depende de la colocación de las otras bacterias, y si depende, de qué manera.

Como se ha mencionado, en la realidad, la posición y rotación exactas de un individuo depende en mayor o menor medida del conjunto total de individuos de la colonia, por lo que f dependería linealmente del número de bacterias, es decir $O(n)$. Por lo tanto, si queremos un orden global $O(n)$, la solución debe disminuir el orden de f a $O(1)$. ¿Es esto posible?

El algoritmo usado actualmente consigue que f se ejecute con O(n), tratando de calcular la física de una bacteria con todas las demás. En el estudio de la matemática de dicho algoritmo se vio que la acción de una bacteria sobre otra, dependía totalmente de la distancia a la que ésta estuviera, siguiendo una distribución normal conforme nos alejamos de ella. Teniendo esto en cuenta, se puede ir más allá y seleccionar un radio constante medido en individuos, y que a cada bacteria sólo le afectase la física de las bacterias a dicho radio de distancia. De esa manera, podríamos reducir en un principio sin perder gran calidad, el orden de f a $O(1)$, dejando que la fiabilidad con la física real lo gobierne la anchura del radio mencionado, siendo éste constante durante toda la ejecución de la simulación.

Además si nos fijamos detenidamente, la dependencia de cuadrática del algoritmo actual viene a partir de que un espacio en la colonia es generado a partir de la resolución de todos los solapamientos internos. Por tanto, si somos capaces de generar dicho espacio de manera manual, reduciríamos esa dependencia consiguiendo que cada resolución se desplace hacia un lugar con relativo espacio libre, veamos cómo hacer esto.

4.2. Expansión por anillos

El algoritmo aquí descrito trata de simplificar el problema de $O(n^2)$ a $O(n)$ perdiendo la menor precisión posible respecto a la realidad en el proceso. Cabe resaltar que dicho algoritmo se aplica en cada fotograma, al igual que se hace con el utilizado actualmente.

4.2.1. Funcionamiento

El método se basa principalmente en generar espacio en la colonia de manera dirigida, y con ello conseguir reducir en π su orden actual. Ese espacio creado artificialmente ha de situarse de tal manera que el solapamiento que allí se desplace provenga siempre de una zona de mayor presión, es decir, de aquella cuyos solapamientos tarden más en llegar al extremo de la colonia.

Para generar dicho espacio, necesitamos saber dónde está el centro de mayor presión de la colonia y a partir de ahí redirigir el solapamiento originado. El lugar de este centro no lo conocemos, pero se puede calcular a partir de un lugar que si podemos determinar en una primera instancia: el borde.

El procedimiento general es el siguiente. Desde el borde se van creando anillos concéntricos hacia el interior de la colonia con una determinada anchura que a partir de ahora llamaremos w, medida en individuos, y que será constante durante toda la simulación independientemente del número de éstos que haya. El último anillo generado, que será el único que no tenga forma de anillo como tal, será el lugar de mayor presión de la colonia.

Una vez tengamos este centro, sabremos desde dónde hay que empujar, y por tanto hacia qué sentido hay que resolver el solapamiento. Empezando desde el anillo más interno (mayor presión) hasta el borde (presión 0), el algoritmo es el siguiente:

1. Relajar el anillo con presión i como si de una colonia independiente se tratara. Para ello, se toma el anillo con presión $i + 1$ como un muro, y el anillo $i - 1$ como si no existiese. Ya que la anchura de los anillos es siempre constante e igual en todos ellos, el número de iteraciones necesarias para realizar este paso es constante también, por lo que es $O(n_i)$ siendo n_i el número de individuos del anillo con presión i.
2. Una vez relajado el anillo con presión i, el radio de éste se habrá incrementado, por lo que hay que empujar el anillo $i - 1$ hacia el exterior, y situarlo nuevamente alrededor suyo. Este algoritmo, como veremos más detalladamente en la sección Etapas, es también de $O(n_i)$

Podemos observar que la idea principal consiste en crear bordes temporales cada w de distancia en la dirección en la que está el borde real de la colonia para conseguir desplazar todo el solapamiento rápidamente, y hacia el lugar que nos interesa. Al relajar un anillo, este se expande únicamente por el lado que más cerca está del exterior de la colonia. Como hemos ido creando anillos desde el borde, cada uno de estos tiene una

morfología aproximada al borde real de la colonia, y en su expansión siempre se desplazará hacia dicho borde.

El orden algorítmico de esta solución es $O(n)$, ya que la fase de detección de bordes es $O(n)$, la de anillado $O(n)$, y tanto el algoritmo de relajación como el de expansión de todos los anillos son $\sum O(n_i) = O(n)$. Esto se podrá ver más detalladamente en el apartado de Etapas.

Esta ganancia no es totalmente gratis. Al crear espacio de manera artificial estamos rompiendo vecindades. Eso significa que la física de las bacterias en la frontera externa a un anillo se realiza sólo con las bacterias interiores, y no con las exteriores, perdiendo fiabilidad con la física real. Esto provoca una dependencia de la calidad respecto a w, ya que a mayor anchura de anillo, menos anillos y menos fronteras. En el apartado de Calidad/Velocidad se explica con mayor profundidad ésta dependencia.

4.2.2. Etapas

A continuación se va a explicar cada etapa más detalladamente, profundizado en las características más importantes que estas tienen para el buen funcionamiento global.

4.2.2.1. Etapa de detección de bordes

El objetivo de esta etapa es determinar el borde de la colonia. Es relevante destacar que esta podría tener cualquier morfología o incluso podría estar dividida en el espacio de simulación, por lo que el algoritmo escogido a la hora de implementarlo ha de ser capaz de funcionar bien en tales circunstancias.

El borde se podría ver como el primer anillo creado, el cual tendrá presión 0 (ausencia de presión). Aunque como se ha visto la calidad del algoritmo depende de w, la anchura de este anillo puede ser menor que la anchura media de todos los demás anillos sin que ello repercuta a la calidad del sistema. Esto es así ya que el anillo con presión 0 es el único anillo cuya frontera es real.

El algoritmo que determine la propiedad de si una bacteria es o no perteneciente al borde, debe asegurar que dicha bacteria no pertenece a la parte interna de la colonia (a no ser, claro está, que haya un espacio vacío que pueda ser considerado un borde como tal). Un error en dicha detección provocaría un fallo en el funcionamiento global del algoritmo. Este fallo ocurre porque el desplazamiento de gran parte del solapamiento de la colonia desembocará hacia el lugar detectado falsamente, el cual sería incapaz de albergar tanta superficie, generando solapamientos abruptos, y por lo tanto creando disparidades con la física real.

Sin embargo el algoritmo no tiene por qué ser capaz de determinar el 100% de bacterias situadas en el borde. Tan solo para porcentajes muy bajos de no detección, generaría una expansión un tanto irregular de la colonia. Esto es así, porque una bacteria considerada borde, a efectos visuales, provoca un “estiramiento” de la colonia hacia ella. Si la colocación de estas es muy dispar o hay extremadamente pocas, podrían deformar la morfología externa de la colonia.

Esta etapa consiste, a lo sumo, en recorrer todas las bacterias de la colonia para calcular una propiedad de ellas, que es dependiente como mucho de las vecinas a un radio constante. Por lo puede ser computada con un $O(n)$. Para una implementación de esta etapa, ver Anexo B, Algoritmos para la implementación de las etapas.

4.2.2.2. Etapa de anillado

Consiste en encontrar el centro de mayor presión, al que también llamaremos anillo con presión m. El algoritmo que resuelva esta etapa ha de ir creando anillos concéntricos, cada uno a partir del anterior, desde el anillo con presión 0 o borde hasta haber recorrido a toda la población.

Todos los anillos creados han de tener la misma anchura w (medida en individuos) que además es constante durante toda la simulación. La anchura de los anillos permite realizar la física en grupos de bacterias, ya que la dependencia física entre las bacterias cercanas es muy grande. A mayor anchura conseguiremos mejores resultados, además de reducir el número de fronteras entre anillos, que como se explicó, repercute en cierta medida en la física de las bacterias.

Aunque esto a priori parece un problema de relativa importancia, se puede paliar en gran medida con un proceso de suavización, que consiste en impedir que las fronteras caigan consecutivamente sobre las mismas bacterias. Esto se puede conseguir creando una especie de *offset* para los demás anillos variando la anchura del borde en cada iteración (nunca por encima de w), ya que como hemos visto, el borde puede ser variable sin afectar a la calidad del sistema.

El último anillo computado de presión m, no tendrá forma de anillo, sino más bien de círculo, y siempre y cuando la colonia sea aproximadamente circular (estadísticamente, conforme aumenta el número de bacterias tiende a ser más circular). Para colonias no circulares, incluso con formas cóncavas, el centro no tiene por qué ser circular y ni siquiera único. Visualmente, el efecto producido se puede asemejar a un mapa montañoso, en el que vienen indicadas las alturas con diferentes colores. Esta configuración no supone ningún problema añadido si se sigue una correcta implementación de todas las etapas.

A continuación, en la figura 12, se puede apreciar una imagen que muestra los anillos computados por el motor CellEngine.

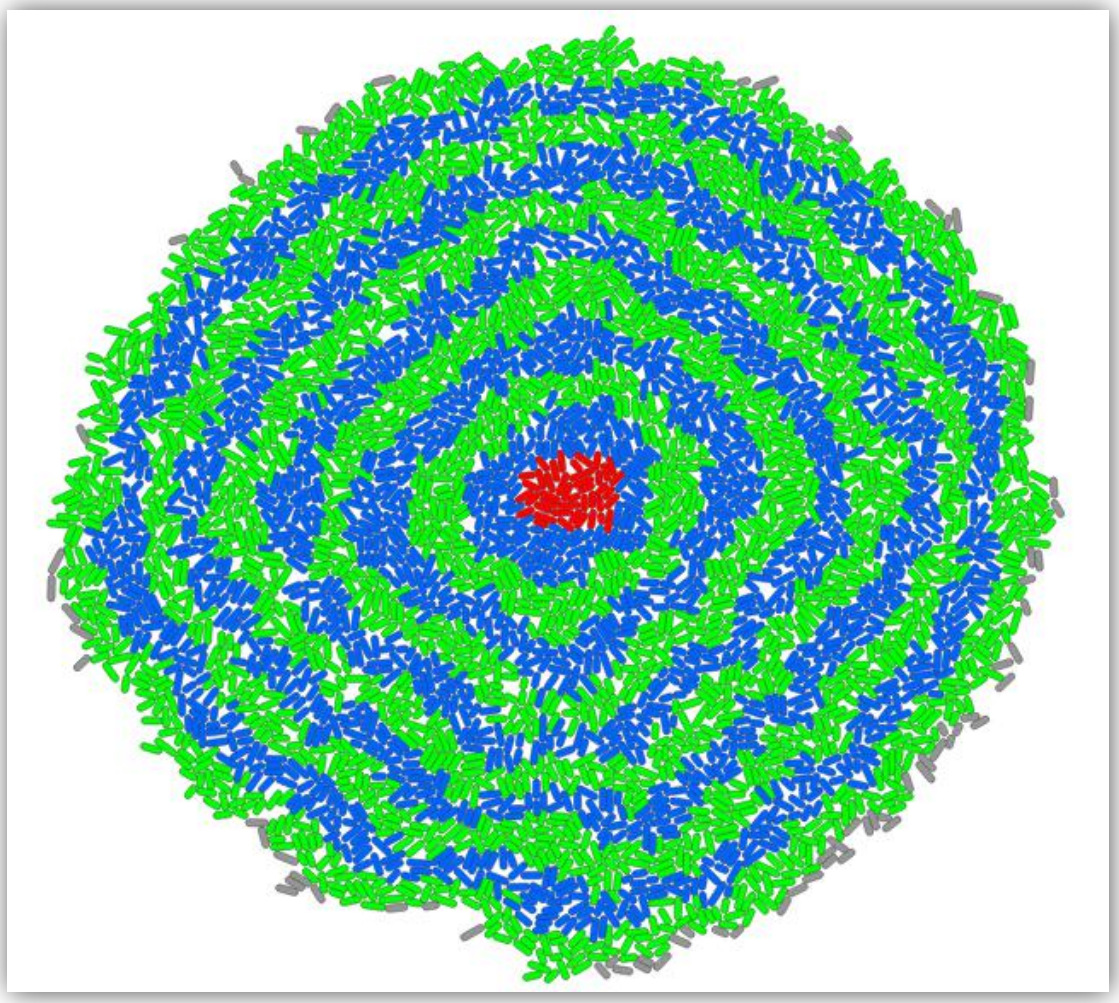

Figura 12. Algoritmo de anillado de CellEngine

Las bacterias grises corresponden al anillo de presión 0 o borde, las bacterias rojas corresponden al anillo de presión m o centro, las bacterias azules son los anillos con presión par y las verdes con presión impar. Es posible pensar que el número de bacterias marcadas como borde es muy reducido y podrían deformar la colonia, sin embargo esto no es así, ya que en cada fotograma se eligen bacterias distintas, regulando el crecimiento morfológico de la colonia.

Para la creación de los anillos se necesita recorrer la población una vez, por lo que el orden algorítmico es lineal respecto al número de bacterias $O(n)$. Varias ideas para la implementación de esta etapa vienen descritas en el Anexo B, Algoritmos para la implementación de las etapas.

4.2.2.3. Etapa de relajación

Esta etapa es la encargada de distribuir el solapamiento hacia el exterior del anillo, y se realizará para todos los anillos. El funcionamiento es idéntico al utilizado actualmente para relajar una colonia entera, con unas diferencias:

- El anillo con presión $i + 1$ (más cercano al centro) se considera estático, es decir se colisiona como si las bacterias de éste estuvieran ancladas.

- El anillo con presión $i - 1$(más cercano al borde) se trata como si no estuviese. La etapa de empuje tratara de solucionar los solapamientos que el anillo i provoque al ocupar espacio perteneciente al anillo de presión $i - 1$.

- El radio de lo que antes era la colonia, ahora es la anchura del anillo w, con lo que se reduce drásticamente el número de iteraciones necesarias para su relajación. Además como dicha w se mantiene constante durante toda la simulación, el número de iteraciones para mantener los anillos relajados también se mantiene constante.

Como cada anillo contiene una determinada porción de los individuos de la colonia, el orden para relajar un anillo, será $O(k \cdot n_i)$, siendo k el número de iteraciones necesarias para la relajación de un anillo de anchura w.

Cada anillo se considera un colonia individual en términos de cuántas iteraciones son necesarias para su relajación, y por lo tanto, se aplica la proporcionalidad de k respecto a r, de la siguiente forma $k = aw^2$, con diferencia de que, lo que antes era el radio de la colonia, ahora es la anchura de un anillo w (medido en individuos) y siendo a, un parámetro de calidad añadido con el fin de reducir el nivel constante de solape.

Como la anchura de w es la misma para todos los anillos, k tendrá también un valor también constante para la resolución de todos ellos. Cabe señalar que aunque los anillos más externos de la colonia contienen más individuos, el número de iteraciones para la relajación de todos es la misma, ya que como hemos visto, k depende de la anchura del anillo.

Ya que k es constante, no afecta a la linealidad del algoritmo. Por tanto, la etapa de relajación se realizará a todos los anillos de la colonia en $\sum O(n_i) = O(n)$, siendo n_i el número de bacterias del anillo i.

4.2.2.4. Etapa de empuje

El objetivo de esta etapa es conseguir un espacio en la colonia que a priori no tenemos. Consiste en una vez relajado el anillo $i + 1$, recolocar el anillo i (más externo) alrededor suyo. Esto es necesario porque tras resolver la fase de relajación al anillo $i + 1$ este se expande solapando al anillo i.

Es posible pensar que el ligero solapamiento que pudiera ocurrir durante un fotograma entre dos anillos se pueda resolver fácilmente en la siguiente iteración. El problema

radica en que el solapamiento se va acumulando anillo tras anillo, de manera que cuando estemos en el anillo de presión 1, solaparemos al anillo de presión 0 con aproximadamente $n_r \cdot r_d$ solapamiento, siendo n_r el número de anillos y r_d el solapamiento producido entre los dos anillos más internos de la colonia. Para colonias relativamente grandes, dicho solapamiento producido en los anillos más externos puede ser incluso mayor que el grosor de una bacteria, generando errores graves cuando se realice la etapa de relajación, al crearse un solapamiento por el otro lado ésta.

Por lo tanto hay que adaptar el anillo exterior. La forma de adaptarlo no tiene porqué ser totalmente exacta, ya que después se realizará el paso de relajación sobre el anillo, rellenando cualquier pequeño espacio que se haya podido formar.

Es importante señalar que al empujar el anillo hacia afuera, este anillo pasa a estar en una zona con un área ligeramente mayor de la que tenía antes. Esto para anillos internos no es un problema, ya que esa área es rellenada con el propio solapamiento que el anillo contiene. En cambio para anillos externos y en colonias muy grandes, el empuje ya es considerable y el anillo puede ganar más área del que el solapamiento que éste contiene pudiera rellenar. A primera vista, no parece un problema que afecte seriamente a la colonia, pero estamos generando un espacio entre las bacterias que en la realidad no hay, y por lo tanto nos alejamos de esta. Además ese error se va a ir sumando, y pudiera llegar a generar huecos lo suficientemente grandes como para que el algoritmo de detección de bordes los encuentre y genere un borde donde no debiera haberlo.

La forma de paliar esto es comprimiendo la anchura del anillo. No es necesario determinar el valor de compresión ideal, simplemente basta con asegurar que no habrá más espacio del que debiera, ya que en la etapa de relajación se solucionará el solapamiento. Se pudiera pensar que al comprimir la anchura del anillo estamos creando una discrepancia de anchuras entre anillos a lo largo del radio de la colonia, sin embargo no es un error en sí, ya que mientras ningún anillo supere el w establecido, el algoritmo de relajación será capaz de relajarlo sin problemas. Además este fenómeno no provoca ningún efecto en cadena en el que cada vez los anillos exteriores sean más delgados, ya que en la siguiente iteración se volverán a computar nuevamente.

El algoritmo de empuje se aplica a todos los anillos exceptuando el de mayor presión (centro de la colonia) y tiene que empujar n_i bacterias por cada anillo. Esto se puede computar en $O(n_i)$, como se puede apreciar en el algoritmo descrito en el Anexo C: Algoritmos para la implementación de las etapas. Por tanto, todos los individuos de la colonia pueden ser empujados con $\sum O(n_i) = O(n)$

4.2.3. Calidad/Velocidad

Ya hemos visto que el número de iteraciones es cuadráticamente proporcional al radio, y en su defecto a la anchura del anillo w. Veamos ahora cómo varía esa proporcionalidad y de qué depende.

El ratio calidad/velocidad (entendiendo por calidad la fiabilidad con la realidad) se basa principalmente en un equilibrio entre estos dos sucesos:

- Generar espacio en la colonia para las resoluciones de solapamiento, y de esa manera no tener que esperar a que dicho solapamiento llegue poco a poco hasta una zona con espacio libre.
- Computar la física de una bacteria teniendo en cuenta a las bacterias cercanas a una determinada distancia de ésta. Recordemos que la posición y orientación dependen sobretodo de las fuerzas realizadas por las bacterias más cercanas.

Ambos sucesos se controlan con un único valor, la anchura del anillo w. Ésta es proporcional a la calidad e inversamente proporcional a la velocidad de computación, es decir $w = calidad/velocidad$.

Si aumentamos la cantidad de espacio libre ganado reduciendo la anchura de los anillos, necesitaremos menos iteraciones para su computación ya que los solapamientos llegarán antes al extremo del anillo, con un considerable aumento de velocidad de ejecución. A cambio, reduciendo la anchura del anillo disminuimos el radio de vecindad con la que se computa cada bacteria e incrementamos el número de fronteras entre anillos, alejándonos de la realidad física.

Recordemos que como cada anillo se considera una colonia individual en términos de cuántas iteraciones son necesarias para su relajación, el número de iteraciones necesarias será de $k = aw^2$, w medido en individuos. El parámetro a de calidad depende del nivel de solapamiento continuo que se quiere permitir en el anillo, y es independiente de la anchura de éste, como se puede apreciar. Ya que todos los anillos tienen la misma anchura, la cantidad de iteraciones para relajar la colonia al completo, será de k.

Es importante destacar que anillos de 1 de anchura provocan que en la colonia no se genere ni una sola fuerza de oposición al desplazamiento, alejándose considerablemente del suceso real, por lo que mínimo se recomienda anillos de 2 de anchura.

5. Resultados y comparativas

5.1. Rendimiento conseguido

Como hemos analizado, se consigue reducir en n el orden algorítmico, pudiendo ejecutar el algoritmo de relajación en O(n). Hay que tener en cuenta que como n crece extremadamente rápido, una reducción de este tipo alarga mucho nuestro límite de simulación, aquel en el que el tiempo de simulación es mayor que el real.

En la figura 13 se puede observar visualmente el rendimiento conseguido, en cuanto a iteraciones de física para la relajación de la colonia se refiere. Es verdad que aunque el número de iteraciones necesarias para relajarlo es bastante bajo, a ello hay que sumarle la carga producida por las distintas etapas del algoritmo necesarias para poder relajar la colonia con ese k conseguido. Sin embargo, como hemos podido analizar, todas esas etapas son lineales, por lo que la carga producida simplemente equivaldría a sumar a la carga de k una valor constante dado por dicha carga, e independiente de esa misma k. El único problema que podría haber es que la solución actual funcionase mejor durante los primeros segundos de la simulación.

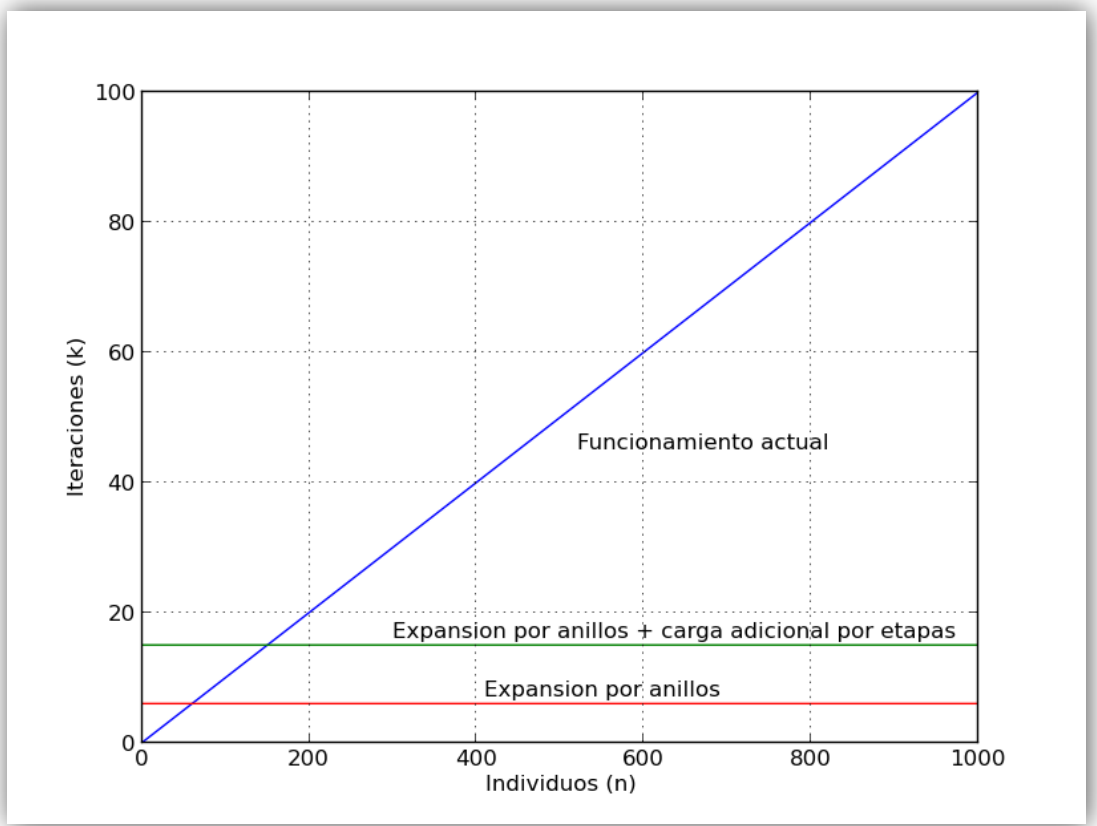

Figura 13. Número de iteraciones en relación al número de individuos.

Esa constancia a nivel de individuos indica que el cómputo físico crece linealmente respecto a ellos. Eso junto al crecimiento exponencial de la colonia genera aproximadamente un constante crecimiento de los individuos simulados, a tiempo real. Aproximadamente cada segundo de simulación tenemos el mismo incremento de individuos. Esto lo podemos ver claramente en los valores de la figura 14, realizada con el motor CellEngine con los siguientes parámetros: $w = 2.5$ y $k = 3$, empezando con una sola bacteria.

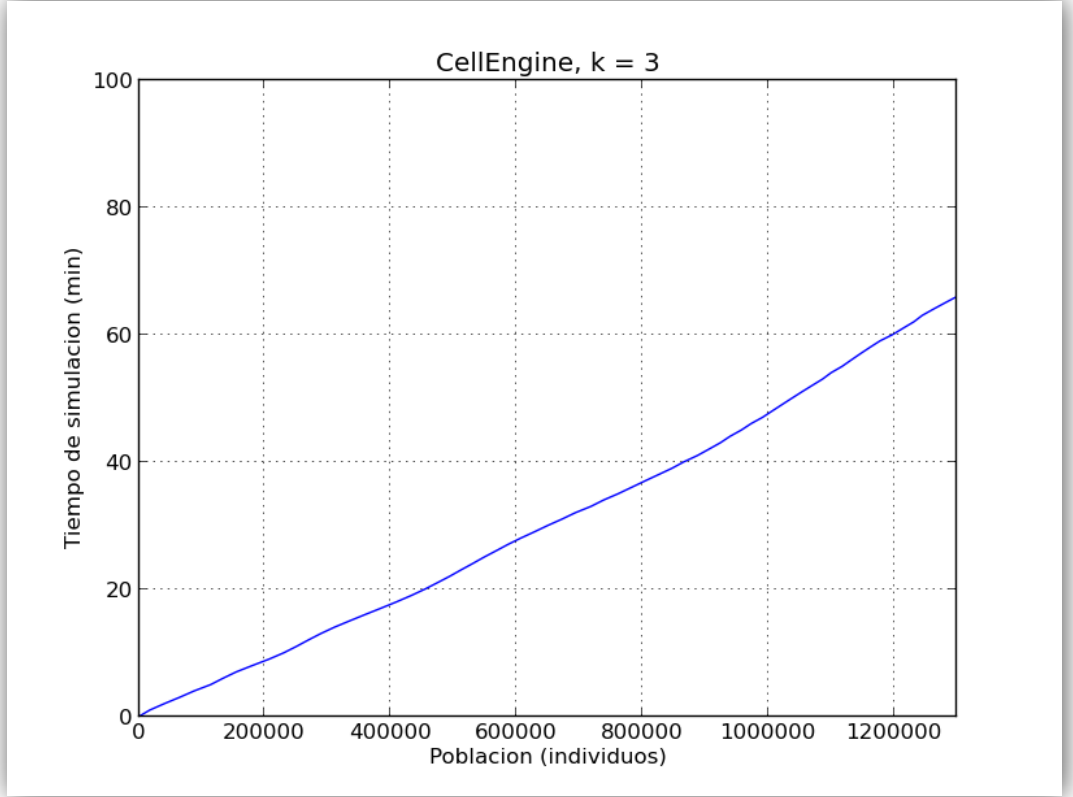

Figura 14. Tiempo de simulación de CellEngine

Si aplicamos la función de porcentaje de solapamiento vista en el planteamiento del problema $S(k, d, r)$, obtenemos un porcentaje de solapamiento para un individuo a distancia $d = 3$ de 0.875 (En el Anexo A: Ejemplo de cálculo de un solapamiento, se explica cómo realizar dicho cálculo). Aparentemente este valor no soluciona casi nada de solapamiento. Sin embargo es suficiente, ya que para el valor utilizado de crecimiento estadístico de las bacterias de la colonia en cada fotograma, conseguimos mantener el solapamiento dentro de unos límites en los cuales apenas puede apreciarse.

Con estos valores podemos calcular cuándo nuestra simulación va más lenta que en la realidad, es decir, cuando el tiempo medio de división en la simulación es mayor que el producido en el mundo real. En este caso, ese valor es de $8.4 \cdot 10^5$ bacterias para una división promedio de 20 minutos (E-Coli). Podemos ver el gran incremento de población que alcanzamos respecto a Gro o CellModeller, en la figura 15. Elegimos estos simuladores por ser del "mismo tipo", que el simulador de pruebas para el motor

CellEngine. Recordemos que estos límites son los alcanzados con un ordenador de sobremesa estándar.

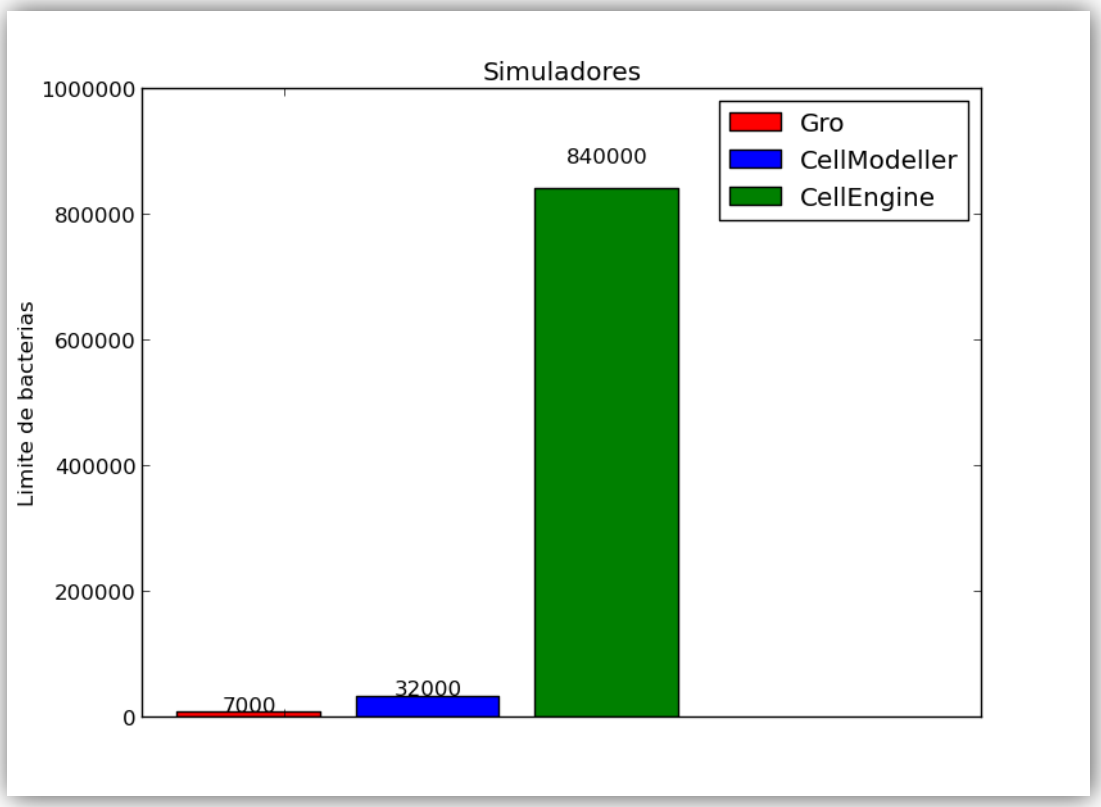

Figura 15. Comparativa de Gro, CellModeller y CellEngine, para cuando la simulación es más lenta que la realidad.

5.2. Validaciones

La forma de determinar si un motor físico orientado a la simulación de colonias de bacterias funciona correctamente es difícil de saber. Muchos de los parámetros que van a determinar si éste se comporta de una manera fiel a la realidad no provienen del motor, sino del simulador (tamaño de bacterias, tipo de disolución, cantidad de nutrientes, etc.). Recordemos que el motor es meramente una pieza de este, y lo que finalmente vemos es la suma de todas esas piezas en su conjunto.

Por otro lado, el algoritmo expuesto aquí no es un motor de físicas en sí mismo, sino una capa extra que se aplica sobre éste y que intenta mejorar el rendimiento en determinadas situaciones. Es por ello que dicha capa no debe afectar (o afectar en lo más mínimo) al funcionamiento que el motor tendría sin ella.

La programación de un motor de físicas no consiste en describir cuál va a ser el funcionamiento de la totalidad de individuos, sino la programación a nivel individual de

éstos. Ello implica que a la hora de programar no se sepa qué efecto provocará en el conjunto la modificación de un pequeño parámetro individual.

Por lo tanto, la manera de saber si lo que se está realizando es correcto es ejecutar el programa y ver qué sucede a continuación. La forma única forma de validación existente en el campo de la biología es la que podemos extraer de la observación de una colonia real, y a partir de ahí, ver que el simulador se comporta lo más "parecido" a ésta.

Hay varias características en los experimentos con colonias de bacterias que se mantienen más o menos constantes independientemente del tipo de experimento que sea:

- La densidad de la colonia en un primer piso de ésta se mantiene constante. Hablamos de un primer piso ya que según nos acercamos al centro, debido a la presión las bacterias, éstas empiezan a traspasar a la tercera dimensión. Para una simulación 2D, se restringe tal efecto, por lo que consideramos una densidad homogénea de bacterias, a lo largo de toda la colonia.

- La forma final de la colonia para un número de bacterias elevado ha de ser circular, ya que conforme aumentemos el tamaño de la población más caos habrá y más equidistante será la distribución de fuerzas. Estudios aseguran que a partir de 30 individuos, la orientación de éstos es totalmente aleatoria [19].

- Como se vio, las bacterias tienden a realizar entre ellas un efecto llamado: efecto neumático. Éste consiste en que las bacterias cercanas entre sí se alinean formando un efecto parecido al encontrado en un neumático. Es por ello que estadísticamente, una bacteria tendría que tener una dirección parecida a la de sus vecinas. Este efecto se puede apreciar en la figura 11, previamente mostrada.

- Para poblaciones considerablemente grandes, se generan formas fractales como los de la figura 16 debido a las fuerzas ocasionadas en los extremos de cúmulos de bacterias alineadas en una misma dirección.

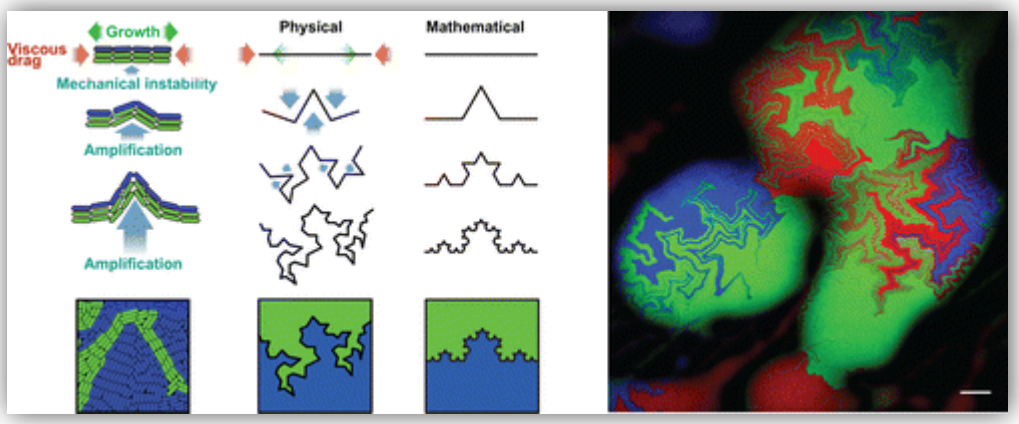

Figura 16. Efecto fractal.

5.3. Pros y contras

5.3.1. Ventajas

Aparte del incremento de rendimiento que ofrece el algoritmo aquí descrito. Otras características relevantes del éste son:

- Se ejecuta por estructura, sólo cuando se necesita. Si no hay suficiente presión en determinadas zonas del espacio, éstas serán marcadas como bordes, y el algoritmo de creación de anillos no se ejecutará ya que no habrá ninguna otra bacteria además de las marcadas con presión 0. Dichas bacterias marcadas como bordes se resolverán en la etapa de relajación sin ser modificadas por la etapa de empuje, por lo que a términos prácticos los resultados son idénticos al algoritmo actual.

- Permite, también por estructura, relajar dos o más colonias que estén en el mismo espacio de simulación. Si el crecimiento de estas produce una colisión entre ellas, el efecto es similar al ocurrido en la realidad, como se puede apreciar en la figura 17, simulada con el motor CellEngine. Cabe destacar que el anillo aparentemente más grande tiene el mismo radio que los demás, ya que la mitad de él es expandido al borde superior y la otra mitad al inferior

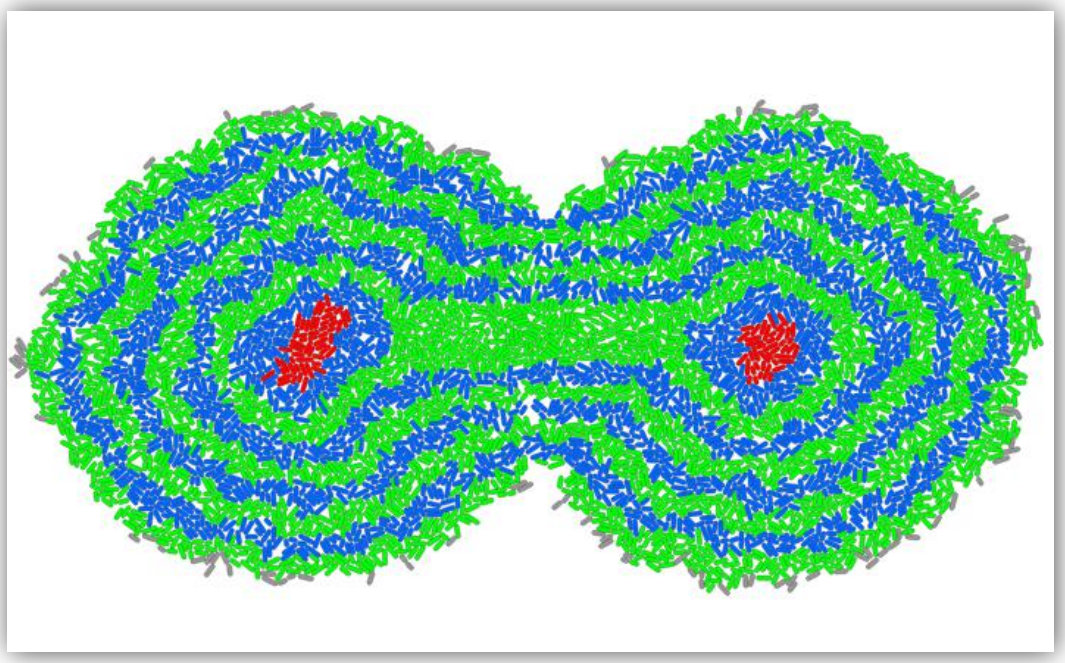

Figura 17. Colisión de colonias mediante el motor CellEngine.

.

5.3.2. Problemas

El problema principal es la pérdida de caos. Como hemos analizado, la posición y rotación de una bacteria depende de las fuerzas de todas las demás bacterias de la colonia. El algoritmo descrito crea una simplificación reduciendo ese radio de acción. Además cada anillo crea una frontera virtual en la colonia, impidiendo las fuerzas que allí se producirían. Se puede observar experimentalmente que conforme reducimos la anchura de los anillos perdemos el ya nombrado efecto neumático, y por consiguiente, parte del efecto fractal formado en poblaciones grandes. Ese grado de calidad que perdemos hace que el algoritmo esté destinado a su uso como herramientas de prototipado.

6. Conclusiones y futuros trabajos

Como hemos visto a lo largo este documento, el problema que se intenta solucionar es la cantidad de cómputo que se necesita para relajar las fuerzas originadas en una colonia de bacterias. La solución actual consiste en aplicar la física que observamos en la realidad y tratar de imitar su funcionamiento lo mejor que se pueda. El problema está en que la replicación de éstas, tal cual, conlleva un orden algorítmico de $O(n^2)$ ya que la situación espacial de una bacteria depende en mayor o menor medida de todas las demás bacterias de la colonia.

Al estudiar el algoritmo actual, cuyos resultados son bastante fieles a la realidad, pudimos observar que la dependencia de unas bacterias con las demás no es lineal conforme nos alejamos de ella. A unas pocas vecindades de distancia, la fuerza que esas vecinas actúan sobre una en concreto es prácticamente nula. También hemos podido analizar el ritmo al que conseguimos resolver el solapamiento originado desde el interior hasta el exterior. Vimos que para un pequeño radio era relativamente rápido, y en cuento incrementaremos ligeramente el valor de dicho radio, las iteraciones necesarias para resolver la misma cantidad de solapamiento crecía demasiado rápido, cuadráticamente.

La solución vendría de combinar ambas características analizadas: tratar de mantener siempre un radio pequeño, y conseguir que una bacteria colisione sólo con las de sus vecindades más próximas. La implementación analizada que cumple con estas dos características consiste en la creación de anillos a lo largo del radio de la colonia. Éstos permiten reducir virtualmente el radio de dicha colonia a un valor muy reducido, y lo más importante, constante. Como se ha visto, se consigue reducir considerablemente el número de iteraciones, llegando a un orden algorítmico de tan solo $O(n)$.

El lado malo de ésta simplificación reside en cada anillo rompe la vecindad de las bacterias por el lado exterior de éste, y por lo tanto perdemos cierto grado de fiabilidad. Aunque también se ha visto que hay varias formas de disipar el error tratando que los bordes no caigan siempre sobre las mismas bacterias.

A parte de esta pequeña discrepancia, el rendimiento es considerablemente alto. Por poner un ejemplo: con una simulación en Gro durante 7 días, 24 horas al día, se consiguen 10^5 individuos; con el algoritmo de Expansión por anillos, conseguimos el mismo número de bacterias en 4 minutos 30 segundos.

Se ha estado hablando durante todo el documento de CellEngine, un motor físico realizado para la implementación del algoritmo (figura 18). El objetivo de éste, aparte de realizar las pruebas de su funcionamiento, es poder integrarlo en un futuro en el simulador de Gro, es decir, sustituir el motor físico de Chipmunk que tiene actualmente este simulador, y cambiarlo por CellEngine. Ello permitiría mantener toda la potencia de simulación que nos ofrece Gro con el rendimiento de CellEngine, que como podemos ver en la figura 15 sería bastante considerable. Además, al motor se le tiene pensado paralelizar mediante el uso de GPUs, con el fin de llegar fácilmente a órdenes de magnitud por encima del millón.

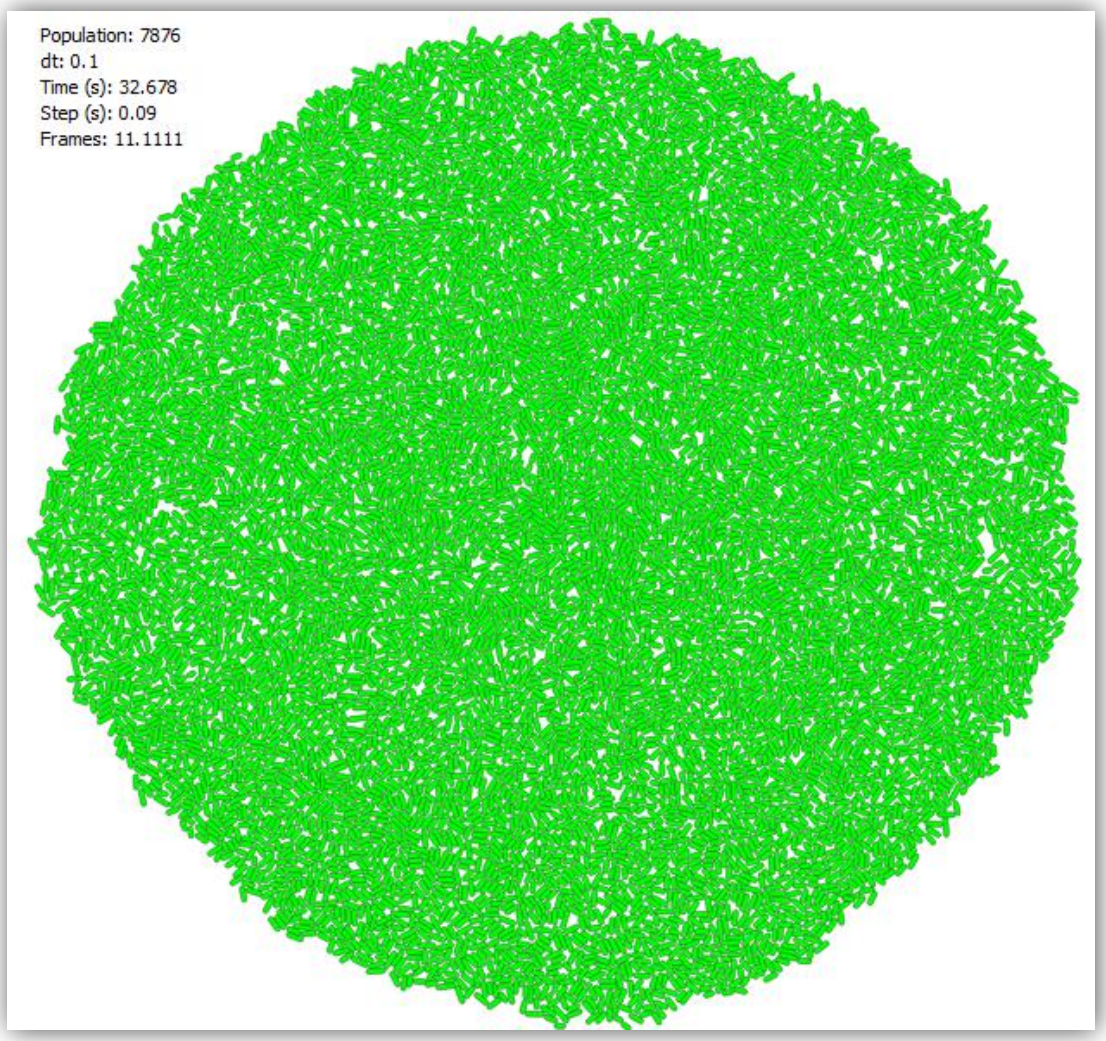

Figura 18. Colonia generada con el motor CellEngine.

También se tiene pensado seguir mejorando los diferentes algoritmos para la implementación de las etapas con el fin de conseguir disminuir el error provocado por las fronteras de los anillos, además de reducir el cómputo lineal añadido por dichas etapas. Otro futuro trabajo, consiste en añadir características como la integración de paredes, con el objetivo de poder usar el motor en simuladores de sistemas microfluídicos.

Referencias

[1] Game Physics Engine Development - Ian Millington

[2] Hoehme, S., and Drasdo, D. (2010) A cell-based simulation software for multi-cellular systems. Bioinformatics 26, 2641−2642.

[3] Andrianantoandro E, Basu S, Karig DK, Weiss R (2006) Synthetic biology: new engineering rules for an emerging discipline. Mol Syst Biol 2: 2006.0028.

[4] Heinemann M, Panke S (2006) Synthetic biology, putting engineering into biology. Bioinformatics 22: 2790–2799.

[5] O'Toole G, Kaplan HB, Kolter R (2000) Biofilm formation as microbial development. Annu Rev Microbiol 54: 49–79.

[6] 4. A bunch of tiny individuals-Individual-based modeling for bacteria IBM Helweber Bucci 2009 FUND

[7] Tesis Máster Antonio García conjugación agosto 2011 FI UPM

[8] Real-Time Collision Detection-Christer Ericson

[9] Jang et al. - Specification and Simulation of Synthetic Multicelled Behaviors - ACS Synthetic Biology - 20128

[10] https://chipmunk-physics.net/

[11] Computational Modeling of Synthetic Microbial Biofilms Andrew Philips Jim Haseloff FUND CellModeller

[12] BSim- An Agent-Based Tool for Modeling Bacterial Populations in Systems and Synthetic Biology Di Bernardo FUND

[13] Lardon et al. - iDynoMiCS next-generation individual-based modelling of biofilms. - Environmental microbiology - 2011

[14] D5.2: Reports on experimental validation and theoretical analysis: Bactosim I: Individual Based Model (IBM) for theoretical analysis of conjugative plasmid propagation

[15] http://www.plaswires.eu/

[16] (23) Cullum, J., and Vicente, M. (1978) Cell growth and length distribution in Escherichia coli. J. Bacteriol. 134, 330−337.

[17] Cho, H., Jönsson, H., Campbell, K., Melke, P., Williams, J. W., Jedynak, B., et al. (2007). Self-organization in high--density bacterial colonies: efficient crowd control.

[18] Volfson, D., Cookson, S., Hasty, J., and Tsimring, L. S. (2008) Biomechanical ordering of dense cell populations. Proc. Natl. Acad. Sci. U.S.A. 105, 15346−15351. (efecto neumático)

[19] Storck Variable Cell Morphology Approach for Individual-Based Modeling of Microbial Communities FUND

Anexo A: Ejemplo de cálculo de un solapamiento

Veamos cómo se puede realizar el cálculo de un solapamiento para una determinada iteración k. Para ello, hay que encontrar todos los $t_{x,k}$ para dicha k. Los valores de $t_{x,k}$ nos permitirán conocer cosas curiosas sobre un solapamiento en concreto, como:

- Cuál es el solapamiento actual respecto al que había inicialmente.
- Cuánta fuerza ejerce sobre sobre el actual otro que haya a x individuos de distancia.
- Cuál ha sido la fuerza media de a favor de la resolución del solapamiento y la fuerza media en contra de éste.

Para realizar estos cálculos necesitamos resolver un Triángulo de Pascal acotado. El procedimiento de resolución de un Triángulo de Pascal normal consiste en sumar los 2 elementos adyacentes de la fila superior de la casilla que se quiere calcular, como se muestra en la figura 19.

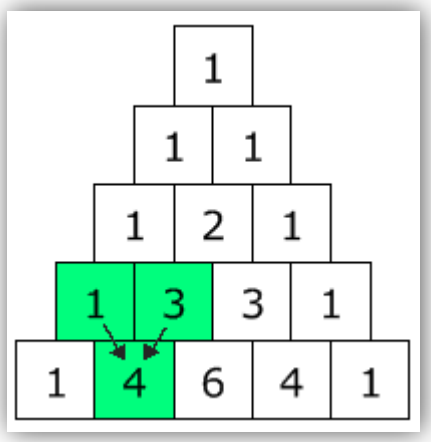

Figura 19. Cálculo de un triángulo de Pascal

Acotar un triángulo de estos consiste en colocar 0, en aquellas casillas que no correspondan a un solapamiento dentro de la colonia.

Supongamos una colonia de 4 individuos de anchura, en la cual queremos saber el porcentaje de solapamiento a 2 individuos del exterior, tras haber aplicado 8 iteraciones de física, consideraremos que inicialmente hay un solapamiento homogéneo. Por tanto tenemos que resolver el triángulo para $k = 8$, $d = 2$ y $r = 4$.

Como tenemos 4 individuos de radio en la colonia. Habrá estadísticamente 3 solapamientos a lo largo del radio. Como $d = 2$, significa que queremos hallar el valor del solapamiento entre el segundo y tercer individuo. El tratamiento de los límites

consiste en poner un 0 a partir de los valores $t_{x,y}$ del triángulo cuya x, no supere la siguiente condición:

- El límite derecho del triángulo será el límite a partir del cual no hay solapamientos en la colonia (fuera de los límites de ésta). y por lo tanto, será a partir de $t_{1,y}$ $\forall y < k$, ya que $d - x < 1$ a partir de $x = 2$.

- El límite izquierdo será el límite contrario de la colonia que estará en $t_{4,y}$ $\forall y < k$ ya que a partir de $x = 4$, $d - x > r - 1$.

El procedimiento para calcular el triángulo se describe en la figura 20.

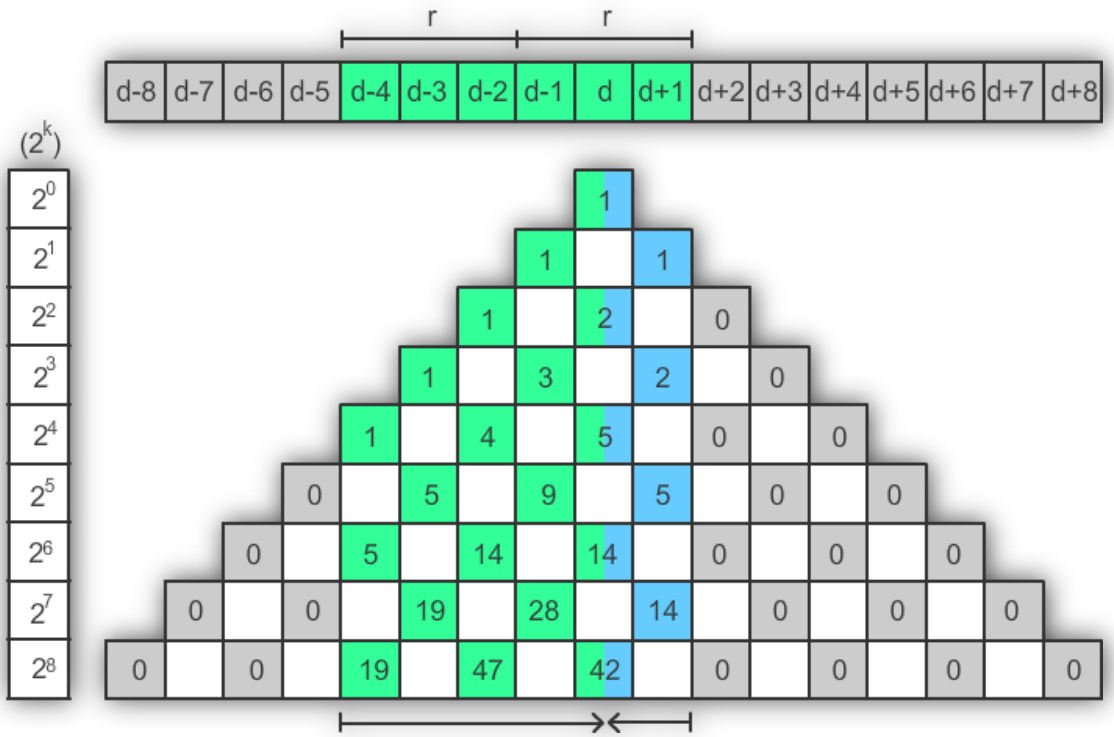

Figura 20. Triángulo de pascal acotado, correspondiente a una colonia con 3 solapamientos a lo largo del radio.

- Recordamos que el porcentaje de solapamiento consiste en aplicar:

$$S(k,d,r) = \frac{t_{k,k} + t_{k-2,k} + \cdots + t_{k-2,k} + t_{k,k}}{2^k}$$

Por tanto $S(8,2,4) = (19 + 47 + 42) / 2^8 = 108 / 2^8 = 0.42$. Después de 8 iteraciones aún falta por resolver un 42% del solapamiento inicial.

- La fuerza que ejerce un solapamiento sobre el actual es:

$$\mathrm{F(d,k)} = \frac{t_{d,k}}{2^k}$$

Por lo tanto, al solapamiento situado en $d - 4$ (al otro lado de la colonia) ejerce una fuerza de empuje hacia el elemento d, en la iteración 8 hacia la derecha de $19/256$, incrementando en esa cantidad el solapamiento inicial que hubiese en d.

- La fuerza media sobre el solapamiento d durante k iteraciones es:

$$\sum_{i=1}^{k} F(x, i)$$

Las fuerzas producidas para valores de x mayores que d son fuerzas de oposición (área azul), y las producidas para valores menores son de empuje (área verde).

Anexo B: Algoritmos para la implementación de las etapas

En este anexo se detallan diferentes algoritmos para implementar las etapas de las que consta el procedimiento de Empuje por anillos. La Etapa de relajación no se explicará, ya que como se comentó, su implementación es prácticamente igual a como se implementan los algoritmos de relajación actuales, que es aplicando un número determinado de iteraciones de física.

Etapa de detección de bordes

Algoritmo 1

La idea consiste en iterar todas las bacterias de la colonia y, dependiendo de la densidad de vecinas a un determinado radio, decidir si pertenece o no a un borde. Para computar este algoritmo necesitamos saber el área media que ocupan las bacterias en la colonia. Para ello necesitamos hallar la media de longitudes, que debería ser constante a lo largo de la ejecución para un determinado tipo de bacteria; y la forma de estas, que por lo general es capsular. Ello nos dará una distribución estadística de cada cuanto hay un centro de bacteria. Esta distribución la utilizaremos para determinar si la densidad de centros en el círculo determinado por el radio es aproximadamente igual a la distribución que nos encontramos en el centro de la colonia, o si es considerablemente distinta, por lo que la bacteria en cuestión pertenecerá al borde.

Podemos asegurar el buen funcionamiento de este algoritmo gracias a que estadísticamente la densidad de bacterias a lo largo de toda la colonia se mantiene. Para paliar el posible error dado por dicha estadística, el radio escogido tiene que ser suficientemente grande, ya que como se explicó, no se puede permitir ni un falso positivo.

Como se comprueba toda la colonia y el radio es constante, este algoritmo es de $O(n)$.

Algoritmo 2

Consiste en trazar una cuadrícula a todo el espacio simulable, y asociar cada bacteria con cada casilla de la cuadrícula si está parcialmente sobre ella. Luego recorriendo las casillas en las que haya al menos una bacteria, saber distinguir qué casilla pertenece al borde de la colonia y cual no. En principio, cuanto más grande sea la casilla mayor garantía estadística sobre el número de bacterias que en ella hay.

Si el tamaño estadístico de las bacterias es constante durante toda la simulación (que debería ser así), y las casillas también mantienen su tamaño, el valor con el que se consideraría a una casilla como borde se mantiene a lo largo de la ejecución y no habría que calcularlo cada vez.

Notemos que a diferencia del algoritmo 1, este algoritmo dice si las bacterias pertenecientes a una casilla son borde, por lo que en este caso lo que recorremos son casillas. El orden algorítmico es $O(n)$, ya que número de casillas es constante respecto al número de individuos.

Algoritmo 3

Es una variación del algoritmo 2 y es el usado actualmente por el motor CellEngine. En este motor para la partición espacial de la física se utiliza una cuadrícula con casillas de una anchura dada por la longitud máxima que puede alcanzar una bacteria. Teniendo ya una cuadrícula y las asociaciones de cada casilla con cada bacteria, se decidió aplicar la lógica del algoritmo 2.

El problema está en que una casilla cuyo lado mide poco más que la longitud media de una bacteria no permite estadísticamente resultados muy fiables, son necesarias casillas de una mayor longitud. Para ello se realiza una modificación que consiste en comprobar también el número de bacterias de las casillas colindantes.

El orden de este algoritmo es $O(n)$, ya que el número de casillas colindantes es constante a cada casilla y el número de casillas depende linealmente del número de bacterias.

Etapa de anillado

En esta etapa, aunque a priori su implementación pueda parecer sencilla, es de gran importancia que el algoritmo que lo implemente cumpla la siguiente condición: la frontera exterior de un anillo ha de ser lo más convexa posible y la frontera interior lo más cóncava posible. El porqué de esto sólo se explica con figuras que no sean circulares, o con distintos tamaños de estas.

Para este tipo de casos se produce un efecto un tanto esporádico pero que genera un error de solapamiento extremo (figuras totalmente solapadas) con su consiguiente brusca resolución. Se produce cuando se da el caso de que haya fuerzas del anillo interior que actúan sobre el anillo exterior que no empujen al objeto del segundo anillo hacia el exterior, por las geometrías del primero. Recordemos que para un anillo interior, el exterior es como si no existiese. Ese efecto se puede ver más claramente en la figura 21. Las distintas correcciones de este error se explicarán en los Algoritmos de corrección 1 y 2.

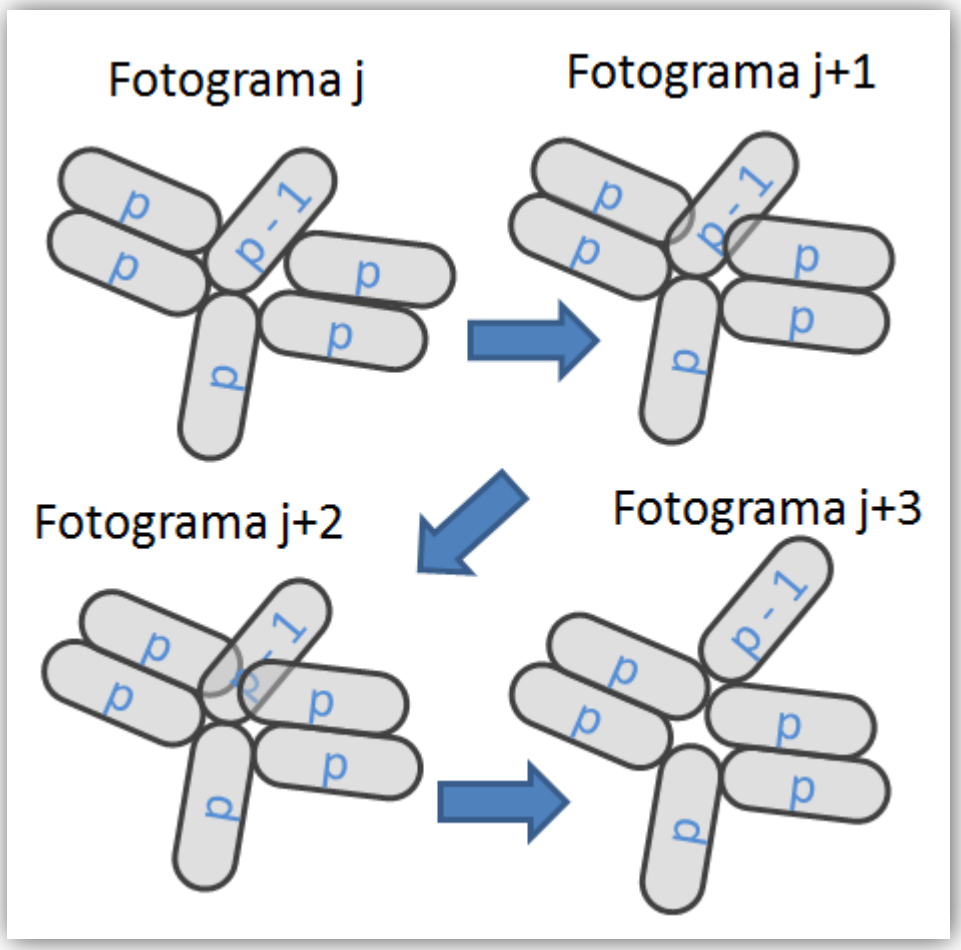

Figura 21. Caso de una bacteria atrapada por bacterias del anillo anterior.

Algoritmo 1

Se basa principalmente en recorrer el anillo anterior, y marcar como "siguiente anillo" a las bacterias que estén a una longitud w de las del anillo a recorrer. Cuando al recorrer un anillo no se encuentre ninguna bacteria. El algoritmo de anillado habrá acabado. Se puede apreciar fácilmente que el orden de este algoritmo es $O(n_i)$, siendo n_i el número de individuos del anillo con presión i. Este algoritmo aplicado a toda la colonia será de $\sum O(n_i) = O(n)$.

Algoritmo 2

Consiste en, mediante los contactos entre las bacterias previamente calculados, realizar anillos de 1 individuo estadístico de anchura hasta llegar al w elegido, la forma de saber hacia dónde crear dichos anillos es seleccionando sólo a los individuos que aún no pertenecen a ninguno de ellos. Este algoritmo empieza por los contactos de las bacterias marcadas como borde o presión 0 y acaba cuando todos los contactos de todas las bacterias ya pertenezcan a un anillo.

Este algoritmo tiene como premisa que todos los individuos internos en la colonia tienen algún contacto durante toda la ejecución. Si esto pudiera no cumplirse, habría que complementarlo con alguna extensión de dicho algoritmo.

Ya que los contactos entre las bacterias no pueden superar un determinado valor, que es constante para todas ellas. Este algoritmo es de orden $O(n_i)$ por anillo, y de $O(n)$ para toda la colonia.

Algoritmo 3

Este procedimiento parte de una cuadrícula, la cual es posible que ya haya sido implementada en la etapa de detección de bordes o incluso antes, como parte interna del motor de físicas. Consiste en marcar las bacterias como "pertenecientes al siguiente anillo" a aquellas cuyas casillas sean colindantes con las del anillo anterior.

La anchura del anillo w, dependerá del número de vecindad utilizado para las casillas colindantes. Este procedimiento puede ser bastante rápido si ya se tiene una cuadrícula de la anchura deseada. Sin embargo no es el procedimiento más escalable, ya que la cuadrícula puede depender de la física y no tener un tamaño fácilmente variable sin afectar a otras partes del motor, y por consiguiente complica el posible juego de w, que tendrá un valor proporcional a la longitud de una casilla.

Como una casilla contiene un número constante de bacterias en su interior, el orden de éste algoritmo también es de $O(n_i)$ por anillo, y de $O(n)$ para toda la colonia.

Actualmente este es el algoritmo utilizado por CellEngine aprovechando que ya contiene una cuadrícula para particionar el espacio físico y que la anchura proporcionada es un valor bastante razonable de calidad/velocidad.

Algoritmo de corrección 1

El primer algoritmo consiste en heurísticas. Depende del algoritmo utilizado, tratar de prever si se puede originar un solapamiento en una determinada zona con una bacteria. Si la previsión es cierta, se marcará a esa bacteria como perteneciente al anillo interior.

Cabe resaltar que es estadístico, y puede no solucionar todos los posibles problemas que pudiese haber. Por lo tanto su uso depende del enfoque al que esté destinado el simulador, es decir, si permite o no un pequeño porcentaje de error.

Dependiendo de qué *inputs* se utilicen para la heurística, éste algoritmo tendrá diversos órdenes. Generalmente si se utiliza en lugares concretos, o en distancias constantes respecto a una bacteria, puede ser fácilmente implementado en $O(n)$.

CellEngine utiliza heurísticas como corrección. Se basa en las casillas de la cuadrícula para determinar cuándo se puede producir el error de que una bacteria se quede "encerrada" o "encerrada" por otras del anillo de mayor presión vecinas de ésta. Para ello se realiza un pequeño cálculo dependiendo de si la bacteria está en parte contenida en casillas con presión mayor o menor que ella. Está implementado en $O(n)$.

Algoritmo de corrección 2

Este método soluciona el 100% de los casos de error aunque es computacionalmente un poco más costoso que el Algoritmo de corrección 1. Consiste en determinar cuándo una bacteria va a ser aplastada en base a las fuerzas que se estén aplicando sobre ella. Hay varias formas de determinar esto:

- Hay una fuerza de gran magnitud, que no pudiese haber sido ocasionada en un único fotograma.
- Actúan sobre ella dos fuerzas medianamente opuestas y con gran magnitud pertenecientes a un anillo inferior.
- En base a los contactos de individuos de anillos inferiores, determinar si geométricamente está rodeada.
- La resolución de su resultante solaparía aún más a una bacteria de un anillo interior.

Con cualquiera de estos cuatro procedimientos se puede determinar si la bacteria va a ser bruscamente solapada, y en tal caso reaccionar marcándola como perteneciente al anillo interior.

Recorriendo todas las bacterias, y teniendo en cuenta que hay un valor máximo de contactos por bacteria, este algoritmo puede ser implementado en $O(n)$.

Etapa de empuje

Algoritmo

Este algoritmo consiste en restaurar el anillo con presión i alrededor del anillo con presión i+1mediante los contactos entre bacterias. La idea consiste en saber la posición de un contacto anterior, para después de la etapa de relajación, determinar dónde debería ir la bacteria con la que antes contactaba.

El procedimiento sería a groso modo, el siguiente, una vez relajado el anillo $i + 1$ por la etapa de relajación:

1. Restaurar mediante los contactos, las posiciones de las bacterias pertenecientes al anillo i y que sean frontera con el anillo $i + 1$.
2. Restaurar las diferentes capas del anillo con un procedimiento similar, en el cual además de moverlo, se aplique la física necesaria para simular un traslado hacia el exterior de la colonia.
3. Aplicar un ligero retraso de movimiento en cada capa del anillo desplazado para evitar conseguir más área del que debiéramos en dicho desplazamiento. Ese

desfase dependerá de la longitud media de recolocación de las bacterias de la primera capa, no de la presión del anillo, ya que ésta última no tiene una correlación directa con el área que dicho anillo pueda ganar.

Para asegurar el funcionamiento de este algoritmo, todas las bacterias tienen que tener algún contacto entre sí. En caso contrario habría que aplicar una lógica adicional al algoritmo que fuese capaz de controlar a una bacteria con ausencia de contactos.

Los contactos tienen un límite máximo por bacteria y se desplaza cada bacteria del anillo una vez. Además, el desplazamiento vendrá determinado por la colocación de la primera capa. Por lo tanto el algoritmo es de $O(n_i)$, por anillo, y de $O(n)$ para toda la colonia.

Anexo C: Función troncal de CellEngine.

A continuación se detalla la función troncal de motor CellEngine, el cual ha sido programado en su totalidad en C++. Esta función es llamada cada fotograma y es la encargada de ejecutar la física y la relajación de la colonia.

```
void CellSpace::step()
{
  /* Delete rings */
  while(!rings.empty())
  {
      delete rings.back();
      rings.pop_back();
  }
  /* Clear spatial table */
  spatialTable.clear();
  /* Clear contacts, stamps, pressures and insert in spatial
table */
  for(std::list<CellBody*>::iterator b = bodies.begin(); b !=
bodies.end(); ++b)
  {
      (*b)->clearContacts();
      (*b)->setLikeMoved(false);
      (*b)->setPressure(NO_PRESSURE);
      (*b)->insertInSpatialTable();
  }
  /* create rings */
  createRings();
  /* Create all contacts */
  createAllContacts();
  /* Expansion algorithm */
  int pressure = rings.size() - 1;
  for(std::vector<std::vector<CellBody*>*>::reverse_iterator r =
rings.rbegin(); r != rings.rend(); ++r)
  {
      /* Push the ring out */
      pushRing(*r, pressure, rings.size() - 1);

      /* Relaxation */
      for(int i = 0; i < 3; i++)
          relaxRing(*r, pressure);
      pressure--;
  }
}
```

Comentarios relevantes

Las partes del código correspondientes a las etapas explicadas son:

- `createRings()`: etapa de detección de bordes y etapa de anillado.
- `pushRing()`: etapa de empuje sobre un anillo, internamente se evita el anillo de máxima presión, el cual no necesita ser empujado.
- `relaxRing()`: etapa de relajación sobre un anillo.

Recorremos todos los anillos desde el centro hasta el borde, y en cada iteración realizamos llamadas a `pushRing()` y `relaxRing()`.

Si nos fijamos detenidamente, la función `relaxRing()`, se llama varias veces, en concreto 3. Este 3 corresponde al número de iteraciones necesarias para relajar el anillo (que hemos llamado *k* a lo largo del documento) hasta un porcentaje de solapamiento que consideramos aceptable.

Para que se entienda el funcionamiento del motor junto con el del simulador, se va a mostrar a continuación el código que llama a la función `step()` en cada fotograma, en este caso perteneciente al simulador de pruebas para *debuggear* CellEngine:

```
      /* Updates cells */
      for(std::list<EColi*>::iterator i = population.begin(); i
!= population.end(); ++i)
          (*i)->update(dt);

      /* Physics step */
      space->step();
```

Como se puede apreciar, en un primer paso se ejecuta el código que controla la acción de cada bacteria `update(dt)`, siendo `dt` el valor de crecimiento asociado a los individuos de la colonia; y en un segundo paso, la física que relajará los solapamientos producidos por la etapa anterior.

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
	Fecha/Hora	Fri Jun 06 19:01:57 CEST 2014
	Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
	Numero de Serie	630
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)